Velos eResearch Design Document

Velos Inc.
2201 Walnut Avenue, Suite 208
Fremont, CA 94538
Phone: (510) 739 4010, FAX: (510) 739 4018

Contents

1. ABOUT VELOS ERESEARCH 4
2. HIGH LEVEL DESIGN (SYSTEM ARCHITECTURE) 4
   - 2.1. APPLICATION ARCHITECTURE 4
     - *2.1.1. Presentation layer 5*
     - *2.1.2. Application layer 5*
     - *2.1.3. The Service layer 5*
     - *2.1.4. The Business (domain) layer 6*
     - *2.1.5. The Persistence layer 6*
     - *2.1.6. Other Java Beans 6*
     - *2.1.7. Data Access Objects 6*
3. HIGH-LEVEL DESIGN CHOICES 7
4. MODULAR DESIGN (LOW LEVEL DESIGN) 10
   - 4.1. STUDY BEAN DESIGN 10
     - *4.1.1. Table Structure 10*
     - *4.1.2. Presentation Layer 12*
     - *4.1.3. Application Layer 12*
     - *4.1.4. Service Layer 14*
     - *4.1.5. Business layer 15*
     - *4.1.6. Helper Classes 16*
   - 4.2. DEPLOYMENT DESCRIPTORS 17
   - 4.3. JAR FILES 18
5. MAIN MODULES 19
   - 5.1. CREATING AN ACCOUNT 19
   - 5.2. LOGIN 20
   - 5.3. ORGANIZATION 21
   - 5.4. GROUP 22
   - 5.5. USER 23
   - 5.6. PERSONALIZE ACCOUNT 24
   - 5.7. EVENT LIBRARY 25
   - 5.8. PROTOCOL CALENDAR LIBRARY 26
   - 5.9. STUDY 28
   - 5.10. PATIENT 29
     - *5.10.1. Patient Enrollment 30*
     - *5.10.2. Patient Schedule 31*
   - 5.11. ALERTS AND NOTIFICATIONS 32
     - *5.11.1. Alerts 32*
     - *5.11.2. Notifications 32*
     - *5.11.3. Options 32*
     - *5.11.4. Adverse Events 33*
   - 5.12. PATIENT CRFS AND OTHER FORMS 34
   - 5.13. BUDGET 35
   - 5.14. MILESTONES 37
   - 5.15. REPORTS 38
     - *5.15.1. Reports Design 38*
     - *5.15.2. Advantage 39*

5.15.3. Reports available in Velos eResearch *39*

6. SECURITY IN VELOS ERESEARCH **41**

7. THE VELOS ERESEARCH MAILER **43**

7.1. HOW IT WORKS 43
7.2. SETTING UP THE ERESEARCH MAILER 44

8. AUDIT TRAIL IN VELOS ERESEARCH **45**

8.1. DESIGN 45
8.2. TABLES 45
8.3. TRIGGERS 46
8.4. PACKAGE AND STORED PROCEDURES 46
8.5. HOW THE AUDIT TRAIL WORKS 46
8.6. AUDIT REPORTS 47

1. About Velos eResearch

Velos eResearch is a comprehensive clinical research management solution designed specifically for investigators and their research teams. It is a web-based application that streamlines and accelerates the research process and helps produce quality study results. Answering the need for a connectivity and data aggregating solution in the field of clinical research, Velos eResearch offers a cost-effective means to bring together all of the participants in the research process. In a secure environment, it provides a platform for collecting and sharing data. This revolutionary new way of managing clinical research improves the efficiency of clinical research, both internally and with trial sponsors. All research-related information is centralized and can be accessed from anywhere, anytime. Password protected, the account is accessible only to authorized personnel. It is easy to use, reliable and completely secure.

2. High Level Design (System Architecture)

The High Level Design is the representation (model) of the software system. This section covers the Architecture of the Velos eResearch application.

2.1. Application Architecture

The Velos eResearch application architecture, (illustrated in Figure 1 below) is a Java 2 platform, Enterprise Edition (J2EE) Layered Architecture. Layered by definition, J2EE is a component-based architecture that separates the presentation content from the underlying application. Layering partitions the functionality of an application into separate layers that are stacked vertically. In layering, each layer interacts only with the layer directly underneath.

These layers can be broadly categorized as:

- The Presentation layer
- The Application layer
- The Service layer
- The Domain layer
- The Persistence layer

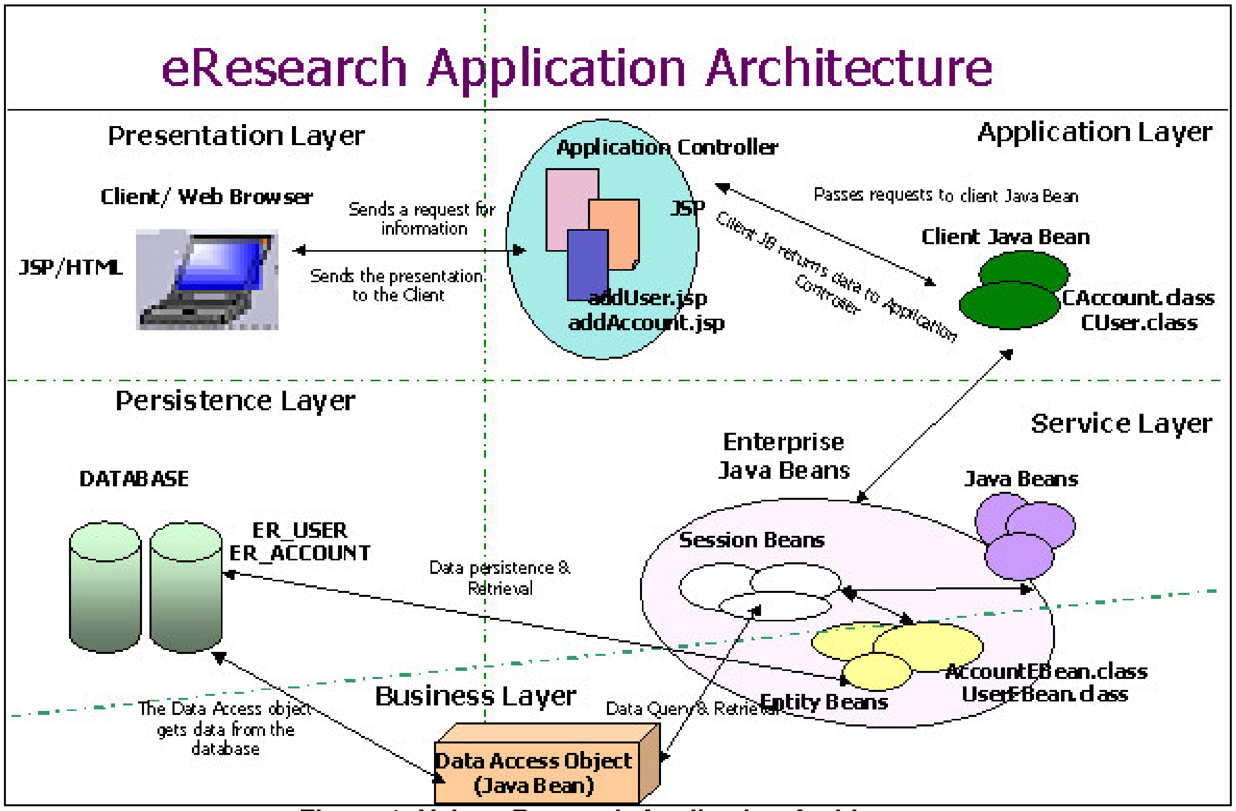


Figure 1: Velos eResearch Application Architecture

2.1.1. Presentation layer

The presentation layer consists of objects defined to accept user input and to display application outputs. The presentation technology used in Velos eResearch is HTML/JSP. This layer interacts with Application Layer only for sending requests and receiving outputs.

2.1.2. Application layer

The Application layer comprises **Controller JSPs** and **Client Java Beans** (Refer to the Architecture diagram in Figure 1). This simplifies the JSP by abstracting out any complex remote method invocation and delegating that responsibility to the JavaBeans. Thus the JSP now concerns itself only with simple logic. It then routes control to various other pages depending on that logic.

2.1.3. The Service layer

The service layer contains controllers and, in the J2EE layer architecture, **Session Beans** represent these controllers. The Velos eResearch application uses the **Session Facade design pattern**, in which the session beans always mask the entity beans. Session beans represent use cases and can therefore delimit transactional boundaries. This object positioning leads to a service-based architecture where the session beans, which represent workflow, control entity beans. Session beans are therefore an extension of the client on the server-side. Session beans come in two flavors: stateless session beans and stateful session beans. The Velos eResearch application has stateless session beans.

2.1.4. The Business (domain) layer

The business layer of Velos eResearch application contains shared business objects, typically shared by multiple clients. The Velos eResearch application implements domain objects as **Entity Beans** offering benefits in distribution, transaction capabilities and persistence. **Entity Beans** represent a row in the database as a Java object (an entity bean) and allows manipulating of that object with public accessor methods.

Entity beans come in two flavors: container manager persistent (CMP) entity beans and bean-managed persistent (BMP) entity beans. In CMP, the basic CRUD (create, read, update, and delete) database operations are delegated to the container. In BMP, the responsibility for writing the SQL code for the database operations is left to the developer. The Velos eResearch application uses CMP entity beans.

2.1.5. The Persistence layer

The Velos eResearch database (Oracle 9i) comprises this layer. A considerable amount of Business Logic is stored in the database in the form of packages, stored procedures, functions and triggers.

2.1.6. Other Java Beans

Some Java Beans embodying generic logic are used in both the Service and the Domain layers. A simple example of a generic Java Bean is a 'FileUpload' bean that can be designed to upload files from the user's system to the Velos eResearch server.

2.1.7. Data Access Objects

The Domain layer includes a few generic Data Access Objects. Stateless session beans interact with these objects to return data for simple queries. These Data Access Objects are simple Java Beans that reduce the load on the Container by eliminating unnecessary Entity Beans. In the Velos eResearch application, Data Access Objects are designed to handle requests for a large number or rows, execute Stored Procedures or database functions and to display Read-Only data.

3. High-Level Design Choices

The Velos eResearch application reflects several design choices that have a global impact on application behavior and performance. To sum up a few:

- **Web-tier business logic vs. enterprise beans components.** Many applications implement all of their business logic in Web-tier classes. More complex and larger-scale applications often choose enterprise beans technology to provide scalability, reliability, a component-based development model and common horizontal services such as persistence, asynchronous communication and declarative transaction and security control. The Velos eResearch application uses **enterprise beans** because of these benefits.

- **Declarative vs. programmatic transaction control**. J2EE transactions can be controlled programmatically in code or declaratively in deployment descriptors. Declarative transactions are easier to create, manage and are more flexible, while programmatic transactions offer a higher degree of control. The Velos eResearch application uses **declarative transaction** control when updating data in the EJB tier.

- **Session Facade vs. directly accessing the data logic.** The Session Facade pattern applies the benefits of the traditional Facade pattern to EJB by completely hiding the object model on the server from the client/presentation layer by having a layer of session beans on the entity beans / other business objects. It not only provides performance benefits, but also suggests a standard architecture for EJB systems-partitioning of the J2EE applications in such a way that the boundary between the client and server is separated by a layer of session beans, whose methods map to (and contain the business logic of) all the use cases in the application. It enforces a highly efficient and reusable design, as well as clearly separates presentation logic (the client), business logic (the session facade) and data logic (entity beans, and so on). The Velos eResearch application uses the **Session Facade** to encapsulate the data logic.

- **StateHolder pattern vs. multiple network calls**. The client tier in an EJB system needs a way to transfer bulk data to and from the server, the design issue being how should a client exchange bulk data with the server without making multiple fine-grained network calls? When a large amount of data needs to be exchanged, this can be achieved by loading many parameters into a method call (while saving) or by making fine-grained calls/remote calls to the server to retrieve data. The former option can quickly get out of hand when dealing with large amounts of parameters and the latter can be a performance killer (executing multiple network calls, requiring the serialization and de-serialization of return values, going through transaction and security checks etc). The Velos eResearch application uses the **Value Object** / **StateHolder** design patterns for exchanging data between various layers of the application. A value object is a serializable Java class that encapsulates business data. To sum up, when the client tier sends data to the EJB tier for processing, the client tier creates a value object to wrap all the necessary information and sends the object to the EJB tier through a Session Façade interface. Likewise, when the client tier receives data from the EJB tier, the EJB tier creates value objects to wrap all information collected from the entity beans, and sends the objects to the client tier through a Session Façade interface. Domain specific value objects are used in this application. A Value Object can reduce network traffic by acting as a caching Proxy for a property of a remote object. As a result, total load on the remote object's remote interface is decreased, because data represented by the Value Object are cached on the client by the Value Object instance. All accesses to the properties of Value Objects are local, and so those accesses cause no remote method invocations. This decreases latency and improves user response time.

- **Synchronous vs. asynchronous communication**. Synchronous communication is extremely useful when an operation produces a result in a reasonable amount of time. Asynchronous communication is more complex to implement and manage but is useful for integrating high-latency, unreliable or parallel operations. Most applications use a combination of synchronous and asynchronous communication. For example, the Velos eResearch application processes a study manipulation request synchronously and generates its complex milestones reports asynchronously.

- **Data Access Objects vs. embedding SQLs**. Having data resource specific code in EJBs/JBs makes a tight coupling between the application and the type of data resource. This can result in an inflexible and difficult-to-maintain application. The design issue is to decouple the EJBs/JBs from the data resource, thus making it easier to change data resource type, data access logic etc. The Velos eResearch application uses the **Data Access Object (DAO)** pattern. This pattern removes the data access specific logic from the EJBs and puts them in a separate interface. As a result, the EJBs/JBs carry on with their business logic and use the Data Access Objects whenever they need to read or modify (only when stored procedures need to be called) the data they represent. This implementation does not couple the EJBs/JBs with their data resource. A change of data resource / data access logic requires only a change in the Data Access Objects. This pattern separates the business logic from the data access logic and makes the application more flexible and maintainable. DAOs in the Velos eResearch application service requests:
    - That may return large number or rows
    - Need to execute a Stored Procedure or a database function
    - Need to display Read-Only data

- **Paged Data Browser display vs. extensive over-running rows of data**. One of the design issues in web-based applications is around presenting a large number of data rows to the user. An application like Velos eResearch may need to display various data browsers/ search results to the user. Barring having a design that displays large and unsophisticated pages with scrollbars, the web client is only capable of presenting a subset of the comlete resultset at any one time and the user may be interested in only a small subset of the entire data. Transmitting the entire resultset as a list of objects to view would be extremely inefficient from the point of view of network traffic and latency. The Velos eResearch application uses the **Paged List**/**Paged Data Browser** pattern to present a paged, scrolling list of data to the user, while minimizing network traffic, server load and latency. Instead of receiving the entire set of rows, the client code accesses **"data by page"**, a sorted list of rows of a given length, starting at a given index. The server-side implementation simply queries the database and returns the appropriate number of items, starting in the appropriate place. The client receives information like total number of rows returned by the query, total number of pages with information, additional pages to be fetched etc. and can present a suitable GUI accordingly.

- **Business Delegate vs. coupling**. This pattern intervenes between a remote business object and its client, adapting the business object's interface to a friendlier interface for the client. It decouples the Web tier presentation logic from the EJB tier by providing a facade or proxy to the EJB tier services. The delegate takes care of lower-level details, such as looking up remote objects and handling remote exceptions, and may perform performance optimizations such as caching data retrieved from remote objects to reduce the number of remote calls. The Velos eResearch application has corresponding **Java Beans** for every session bean. The presentation layer interacts with the Java Bean and is spared from tasks like looking up remote objects and handling remote exceptions. This way the presentation layer is not coupled with the Enterprise beans.

4. Modular Design (Low Level Design)

As mentioned above in the High Level Architecture, the application has an Application layer (comprising of Controller classes and client Java Beans), a Service layer (comprising of Session Beans), a Business layer (comprising of Entity Beans) and a Data Access Object (Java Beans) corresponding to each table in the database. This set of layers is referred to as a set of beans. The architecture is standard throughout the application therefore one set of beans has been selected as a sample for further description. The choice of '**Study Bean**' has been made for the obvious reason that a study is central to the research process.

4.1. Study Bean Design

Description of the Study module has been broken into the following sections for the sake of simplicity and understandability

- Table structure (Domain Layer)
- Presentation Layer
- Application Layer
- Service Layer
- Business layer
- Helper Classes

4.1.1. Table Structure

The study table consists of the following columns

Column name	Description
PK_STUDY	This column uniquely identifies a study. It is the primary Key of the table
FK_ACCOUNT	This column is for storing the id of the account to which the study belongs This id used in case of saving study data for external users. This is a Foreign Key to the ER_ACCOUNT table
STUDY_PUBFLAG	This column is for storing a flag that whether the study should be released for public viewing. Possible Values Y/N for yes / no
STUDY_TITLE	This column is for storing the Title of the Study
STUDY_OBJ	This column is for storing the Primary Objective of the Study
STUDY_SUM	This column is for storing a summary of the Study
STUDY_PRODNAME	This column is for storing the name of the product (in case the study is based on a product)
STUDY_SAMPLSIZE	This column is for storing the sample size of the study
STUDY_DUR	This column is for storing the duration of the study

STUDY_DURUNIT	This column is for storing the Duration Unit of the Study. These units could be Day, Week, Month and Year
STUDY_ESTBEGINDT	This column is for storing the Estimated Begin Date of the Study
STUDY_ACTUALDT	This column is for storing the Actual Begin Date of the study
STUDY_PRINV	Not in use
STUDY_PARTCNTR	This column is for storing the list of Participating Centers of the Study
STUDY_KEYWRDS	This column is for storing the keywords identified for this study. Keywords to be entered for search can match any of the following fields: Phase, Therapeutic Area, Name/Location of Sponsor, Name/Location of Participating Centers, Name of Principal Investigator
STUDY_PUBCOLLST	This column is for storing a comma separated list of columns which the user flags for Public Viewing
STUDY_PARENTID	This column will be used in case of Study Transfer (For External Users). It will store the ID of the original Study. Also, with the help of this column, only the original studies (not the copied ones) will be listed in Study Search results
FK_AUTHOR	This column stores the author/data manager of the study. Stores PK of er_user
FK_CODELST_TAREA	This column is for storing the Therapeutic Area of the Study. This is FK to table ER_CODELST
FK_CODELST_PHASE	This column is for storing the Phase of the Study. This is FK to table ER_CODELST
FK_CODELST_BLIND	This column is for storing the Blinding option of the Study. This is FK to table ER_CODELST
FK_CODELST_RANDOM	This column is for storing the randomization option of the Study
RID	This column is used for the audit trail. The RID uniquely identifies the row in the database
CREATOR	This column stores the name of user inserting the row in the table
STUDY_VER_NO	This column stores the version number of the study
STUDY_CURRENT	Not in use
LAST_MODIFIED_BY	This column stores the name of user who last modified the row in the table
LAST_MODIFIED_DATE	This column stores the date on which the row was last modified
CREATED_ON	This column stores the date on which the row was inserted
STUDY_NUMBER	This is the number given to identify the study by the user
FK_CODELST_RESTYPE	This column stores the Research type e.g. applied, basic, clinical. Stores PK of er_codelst with code type 'research_type'
FK_CODELST_TYPE	This column stores the Study type. Stores PK of er_codelst with code type as 'study_type'
STUDY_INFO	This column stores the information for the study
STUDY_SPONSOR	This column stores the sponsors for the study
STUDY_CONTACT	This column stores the contact persons for the study
IP_ADD	This column is used for Audit Trail. Stores the IP ADDRESS of the client machine
STUDY_VERPARENT	This column stores the study id of the parent study if this study is created as a result of 'create a version' feature
FK_CODELST_CURRENCY	This column stores the currency applicable through out the study. Stores PK of sch_codelst with code type 'currency'

4.1.2. Presentation Layer

The presentation layer consists of two parts – static and dynamic. The dynamic content is built using the JSPs and the static one using HTML pages. The JSPs interact with only the **Application Layer** to fetch any data from the database or application specific settings from the configuration files. For details on this please refer to the 'Various Operations' part at the end of this section.

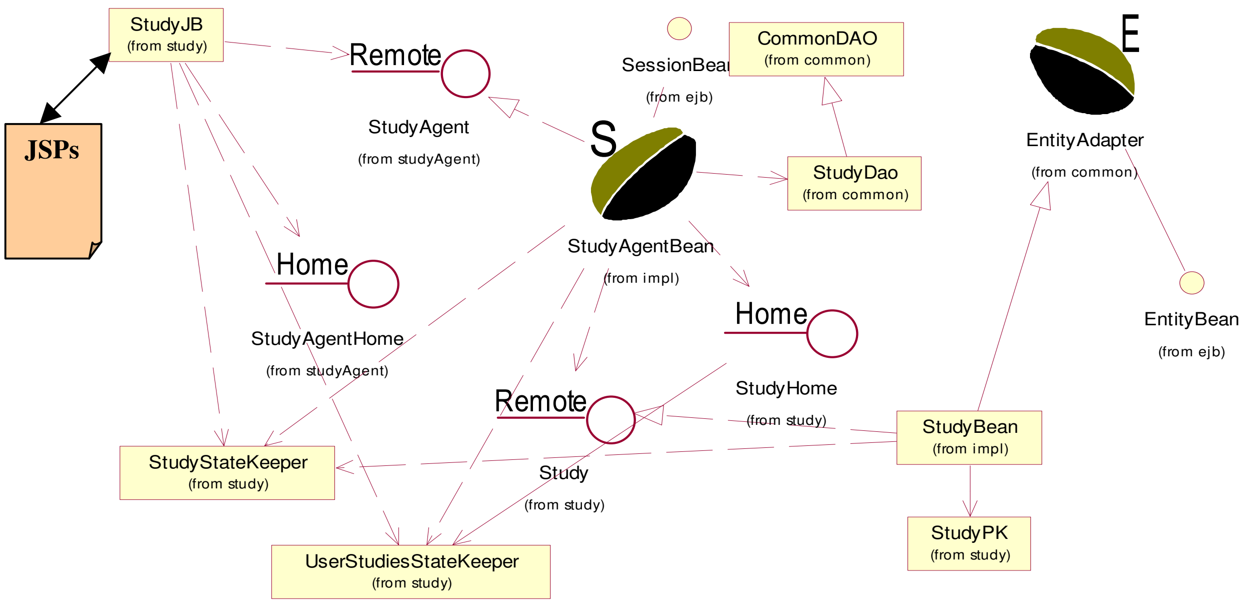


Figure 2: Classes at various layers

4.1.3. Application Layer

The application layer consists of Controller Classes or Client Java Beans. This class is called StudyJB. It is located in the package *com.velos.eres.web.study*. It consists of the following:

- Variables – Each column has a corresponding variable defined in the study table (like id for PK_STUDY, account for FK_ACCOUNT, StudySponsor for STUDY_SPONSOR and so on). This is to ensure that this class is the client side representation of the study object residing at the server/database. These variables are defined as private so that the user does not access these variables directly but rather through the getter and setter methods. By convention, the 'type' for all the variables is kept as 'String' except for the variable corresponding to the Primary Key of the table (PK_STUDY) which is kept as 'int'.

- Getter & Setter (Accessor) methods – A getter and a setter method is defined for every variable. These methods are defined as 'public' and any change to the value of a variable can only be done using these methods. Like getId and setId methods for the variable ‘id’, getAccount and setAccount for the variable ‘account’, and so on.

- Constructors – The constructors defined are –
    - A blank constructor (default)
    - A constructor that takes the primary key as an argument
    - A full argument constructor taking all the variables as parameters

- GetStudyDetails – This method takes no argument, gets the values stored in the database corresponding to the value of the primary key variable (id) and sets the variables of this class with the values stored in the database. This method calls the getStudyDetails method of StudyAgent (a Stateless Session Bean that facades the Study Entity Bean, both are described later in the document) with primary key value as the parameter. It returns a StudyStateKeeper type object containing all the values corresponding to the primary key. These values are then set to StudyJB’s variables. Also the StudyStateKeeper object returned by the StudyAgent object is returned by this method.

- SetStudyDetails – This method is used to add a row of data to the study table. It calls setStudyDetails method of StudyAgent with the StudyStateKeeper as the parameter. The StudyStateKeeper passed as the parameter is populated with the value of the variables before being passed as a parameter. The setStudyDetails method of StudyAgent returns the primary key value created for the row that has been inserted. This value is set to the primary key variable of the StudyJB.

- UpdateStudy – This method is used to modify a row of data in the study table. It calls updateStudy method of StudyAgent with the StudyStateKeeper as the parameter. The StudyStateKeeper passed as the parameter is populated with the value of the variables that needs to be modified before being passed as a parameter. The updateStudy method of StudyAgent returns ‘0’ if the update is successful and ‘-2’ otherwise. This value is returned by updateStudy of StudyJB.

- CreateStudyStateKeeper – This method returns a StudyStateKeeper type object populated with the values of the variables of the StudyJB.

- Module specific methods – Apart from the above-mentioned methods there are some module specific methods used for retrieving the data from the database (mostly Read-Only data).

4.1.4. Service Layer

The service layer consists of Stateless Session Beans. The classes are StudyAgent, StudyAgentBean and StudyAgentHome.

- The ***StudyAgentHome*** is the Home Object of the Study Session Bean (StudyAgentBean). It is located in the package *com.velos.eres.service.studyAgent*. It contains only one method - ‘create’. This method, when called, returns a Remote Object to Study Session Bean (StudyAgent).

- The ***StudyAgent*** is the Remote Interface of the Study Session Bean (StudyAgentBean). It is located in the package *com.velos.eres.service.studyAgent*. It contains the declaration of all the methods contained in the StudyAgentBean. There is no implementation and every method throws a RemoteException.

- The ***StudyAgentBean*** is the Study Session Bean. It is located in the package *com.velos.eres.service.studyAgent.impl*. It consists of the following:
  - Variable – A SessionContext type variable (ctx).
  - SetSessionContext method
  - EjbActivate method
  - EjbPassivate method
  - EjbCreate method
  - EjbRemove method
  - ***GetStudyDetails*** – This method takes an integer as parameter. This is the primary key of the study for which the row is being requested from the database table. This method first calls the findByPrimaryKey on the StudyHome class with the primary key value as the parameter and then calls the getStudyStateKeeper method of the Study class (the remote object of Entity Bean for the study table, described later in the document). It returns a StudyStateKeeper type object containing all the values corresponding to the primary key. This state keeper is then returned by this method.
  - ***SetStudyDetails*** – This method is used to add a row of data in the study table. It takes a StudyStateKeeper type object as a parameter. This object contains the values that are required to be stored in the database. It calls create method of StudyHome (Home object of Study Entity Bean described later in the document). SetStudyDetails returns an integer, which is the primary key of the inserted row.
  - ***UpdateStudy*** – This method is used to modify a row of data in the study table. This method first calls the findByPrimaryKey on the StudyHome class with the primary key value as the parameter and then calls updateStudy method of Study class (the remote object of Entity Bean for the study table, described later in the document) with the StudyStateKeeper as the parameter. The StudyStateKeeper passed as an argument

is the one received as a parameter by this method and contains the value of the variables that needs to be modified. The updateStudy method of StudyAgent returns ‘0’ if the update is successful and ‘-2’ otherwise. This value is returned by this method.

- ***Module specific methods*** – Apart from the above-mentioned methods there are some module specific methods used for retrieving the data from the database. These methods use special classes called ***Data Access Objects*** to fetch Read - Only data from the database. The data access logic is written in ***Data Access Objects*** specific to every module.

4.1.5. Business layer

The Business layer consists of Entity Beans and the classes are Study, StudyBean and StudyHome.

- The **StudyHome** is the Home Object of the Study Entity Bean (StudyBean). It is located in the package *com.velos.eres.business.study*. This contains the following methods:
  - ***Create*** method – This method takes StudyStateKeeper as a parameter and returns a Remote Object to Study Entity Bean (Study).
  - ***FindByPrimaryKey*** method – This method is a finder method for searching a record on the primary key value. It takes a StudyPK type object as a parameter and returns a Remote Object to Study Entity Bean (Study).
  - ***FindByUser*** method – This method is a finder method for searching for studies of a particular user. It takes an integer (userId) as a parameter and returns an Enumeration object containing all the studies of that user.

- The Study is the Remote Interface of the Study Entity Bean (StudyBean). It is located in the package *com.velos.eres.business.study*. It contains the declaration of all the methods contained in the StudyBean. There is no implementation and every method throws a RemoteException.

- The **StudyBean** is the Study Entity Bean that extends *com.velos.eres.business.common.EntityAdapter* (the base class for Entity Beans). StudyBean is located in the package *com.velos.eres.business.study.impl*. It consists of the following:
  - ***Variables –*** A variable is defined corresponding to each column in the study table (like id for PK_STUDY, account for FK_ACCOUNT, studySponsor for STUDY_SPONSOR and so on). These variables are defined as public so that the container can access these variables directly and is able to persist the data. By convention the ‘type’ of all the variables is kept in accordance to the type of the column used to store that variable in the database. For eg int for id, Integer for account, String for studySponsor, java.util.Date for study Estimated begin date etc.

- ***Getter & Setter methods –*** A getter and a setter method is defined for every variable. These methods are defined as 'public' and any change to the value of a variable by the user can only be done using these methods. Like getId and setId methods for the variable 'id', getAccount and setAccount for the variable 'account', and so on.
- ***EjbCreate*** method – This method takes StudyStateKeeper as an argument. It inserts a row in the Study table using the setStudyStateKeeper method and returns a StudyPK object that is the primary key value of the newly inserted row.
- ***GetStudyStateKeeper*** method – This method returns a StudyStateKeeper object with the variables set with the values of the current variables.
- ***SetStudyStateKeeper*** method – This method takes a StudyStateKeeper as an argument with the value of the variables set with the values that are to be set to the columns of the new row. This method generates a primary key using the sequence object for the study table and then sets the variables of the StudyBean class with this and the values in the StudyStateKeeper.
- ***UpdateStudy*** method – This takes a StudyStateKeeper as an argument. This state keeper object contains the values that are to be set instead of the old values. The values are extracted from the state keeper and set to the variables of the Entity Bean. The method return '0' if the operation is successful and '-2' if some exception occurs.
- Blank ejbPostCreate, ejbActivate, ejbLoad, ejbStore, ejbPassivate and ejbRemove methods (inherited from EntityAdapter).
- SetEntityContext and unsetEntityContext methods.

4.1.6. Helper Classes

Apart from the Controller Classes, Service Layer and Business Layer, each module consists of the following helper classes that are used/required by the above layers to perform their task easily and effectively.

- ***StudyPK*** – This is a serializable class located in the package com.velos.eres.business.study and is used by the Entity Bean. It consists of the following variables and methods:
    - Variable 'id' – This is a public variable of type int and denotes the integer value of the primary key column.
    - Constructors – A blank constructor and a constructor that takes the primary key value as a parameter in the form of an integer.
    - ToString method – This method doesn't take any parameter and returns a String that is the primary key value contained in the class.
    - Equals method – this method takes an object as a parameter. The method returns false if the object is null or the object is not of the primary key class i.e. StudyPK. It returns true if

the object is of primary key class type and if all the variables of the class have the same values as the object passed as a parameter to this function.
  - Hashcode method – It returns the integer value of the primary key variable.

- ***StudyStateKeeper*** – This is a serializable class used to pass the variable values from one method to another. It represents the ***StateHolder/Value-Object pattern*** used to transfer data between various layers. It is located in the package com.velos.eres.business.study. It consists of the following:
  - Variables – A variable is defined corresponding to each column in the study table (like id for PK_STUDY, account for FK_ACCOUNT, studySponsor for STUDY_SPONSOR and so on). These variables are defined as private so that the user does not access these variables directly but rather through the getter and setter methods. By convention the ‘type’ of all the variables is kept as ‘String’ except for the variable corresponding to the Primary Key of the table (PK_STUDY) which is kept as ‘int’.
  - Getter & Setter (Accessor) methods – A getter and a setter method is defined for every variable. These methods are defined as ‘public’ and any change to the value of a variable can only be done using these methods. Like getId and setId methods for the variable ‘id’, getAccount and setAccount for the variable ‘account’, and so on.
  - Constructors – A constructor that takes the primary key and the account id as an argument.
  - A full argument constructor taking all the variables as parameters.
  - GetStudyStateKeeper method – This method returns a StudyStateKeeper object with the variables set with values of this statekeeper.
  - SetStudyStateKeeper method – This takes a StudyStateKeeper as an argument and sets the values of the variables of this StateKeeper with the values of the StateKeeper passed as the argument.

4.2. Deployment Descriptors

Deployment descriptors enable EJB containers to provide *implicit middleware services* to enterprise bean components. An implicit middleware service is a service that beans can gain without coding to any middleware API—the beans gain the services automatically. There are 2 deployment descriptor files – ejb-jar.xml and jaws.xml – in each jar file.

- ***ejb-jar.xml*** – This file defines the various ejb objects used. All the beans (session and entity) have an entry in this file. Thus there are 2 (one for session bean and one for entity bean) entries in deployment descriptor of each module. Description of session bean is contained within the elements <session> </session>. This defines the following –

- Name with which the session bean is deployed
- Name of the home object of this bean
- Name of the remote object of this bean
- Name of the EJB class
- Type of the session bean – stateless or stateful. In Velos eResearch only Stateless Session beans are used.

Description of entity bean is contained within the elements <entity> </entity>. This defines the following:

- Name with which the entity bean is deployed
- Name of the home object of this bean
- Name of the remote object of this bean
- Name of the EJB class
- Persistence type of the entity bean – container managed or bean managed. In Velos eResearch only container-managed persistence is used.
- The primary key class of the bean
- Whether or not the bean is re-entrant. In Velos eResearch this is always False.
- An entry for each of the variables defined in the entity bean class corresponding to the database column.
- Apart from the above there are transaction attributes defined in the deployment descriptors. In Velos eResearch the transaction used is 'required'.

- ***jaws.xml*** – Jaws.xml contains the following –
  - Mappings between the java type, the JDBC type and the SQL type for various databases
  - An entry specifying the table corresponding to the entity bean
  - Entries specifying the operations allowed on the table
  - An entry specifying the time-out time between the database and the entity bean
  - Mappings between the table columns and the entity bean variables
  - Entry for the finder method(s)

4.3. Jar files

There is a jar file corresponding to each set of beans (a set of bean consists of controller classes, session bean and entity bean for a table). It is in .zip format and contains the following–

- The Java bean
- The session bean along with the home object and the remote object.
- The entity bean along with the home object and the remote object.
- The state keeper class
- The primary key class

- The CommonDao class
- The deployment descriptors

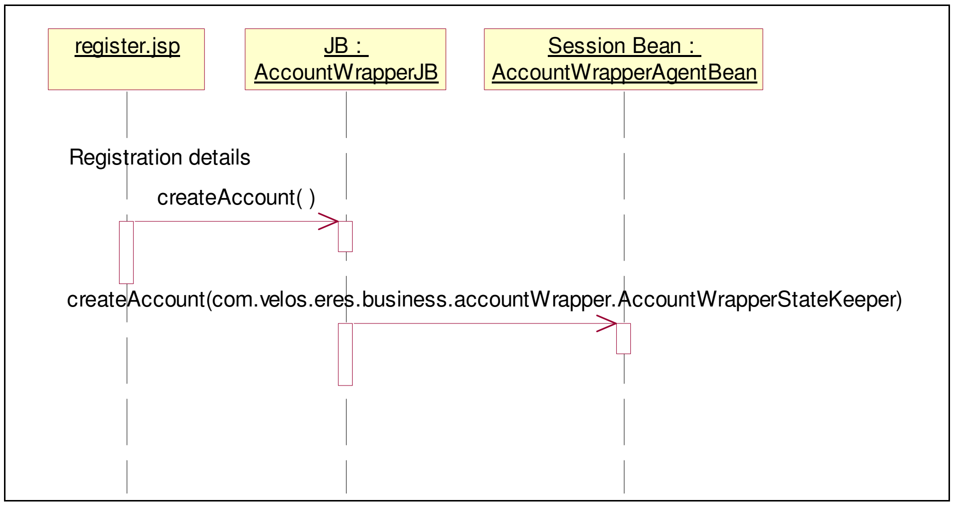
The jar file is located at the path <JBOSS_HOME>\jboss\deploy. Thus there as many jar files in 'deploy' folder as the number of tables in the database. These jar files are used to deploy the beans on the application server. Apart from these jar files there is a '**consolidated.jar**' in Velos eResearch. This jar file contains the contents of all the jar files excluding the deployment descriptors. This jar file is copied at the location <JBOSS_HOME>\jboss\lib\ext.

5. Main Modules

The architecture of all the modules in the application is the same as the study module (explained earlier in the document). A more basic description of each module along with the interaction diagram is given below. The interaction diagrams give a broad overview of JSP interaction with the Java beans, session beans and entity beans (for the sake of simplicity and readability all method calls-object interactions may not be included).

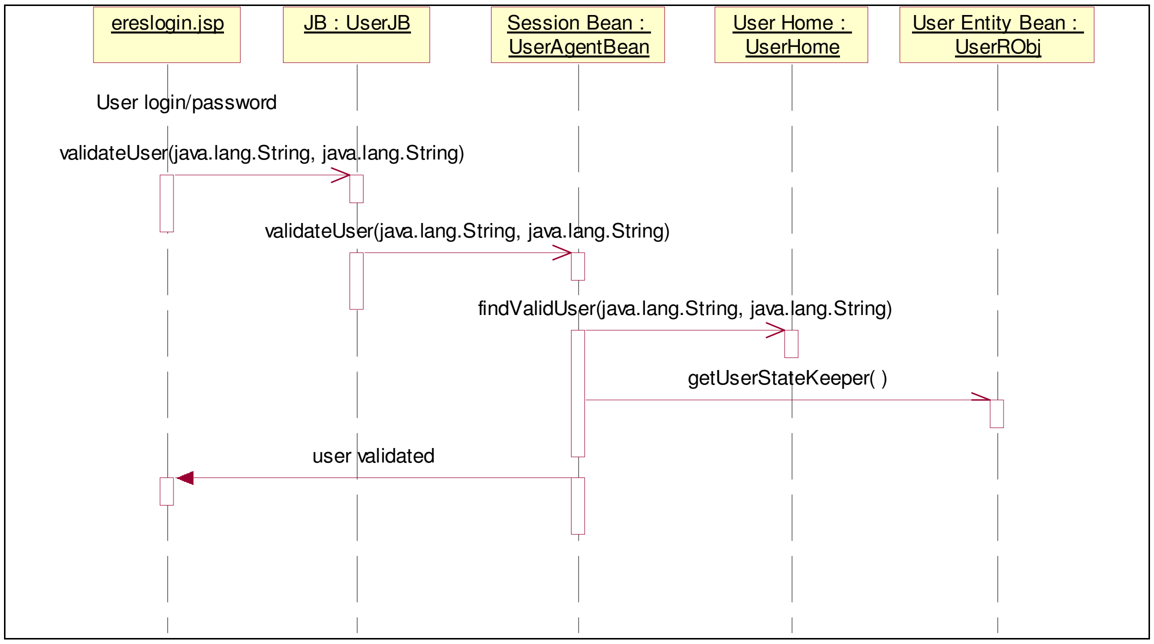
5.1. Creating an Account

The Velos Administrator manages the account creation module. The registration form is completed by the researcher and submitted to the system. This creates an account for him. This account is activated after verification of submitted information by Velos. After authentication, the login and password information is emailed to the user and the account is activated.


register.jsp
JB : AccountWrapperJB
Session Bean : AccountWrapperAgentBean
Registration details
createAccount( )
createAccount(com.velos.eres.business.accountWrapper.AccountWrapperStateKeeper)

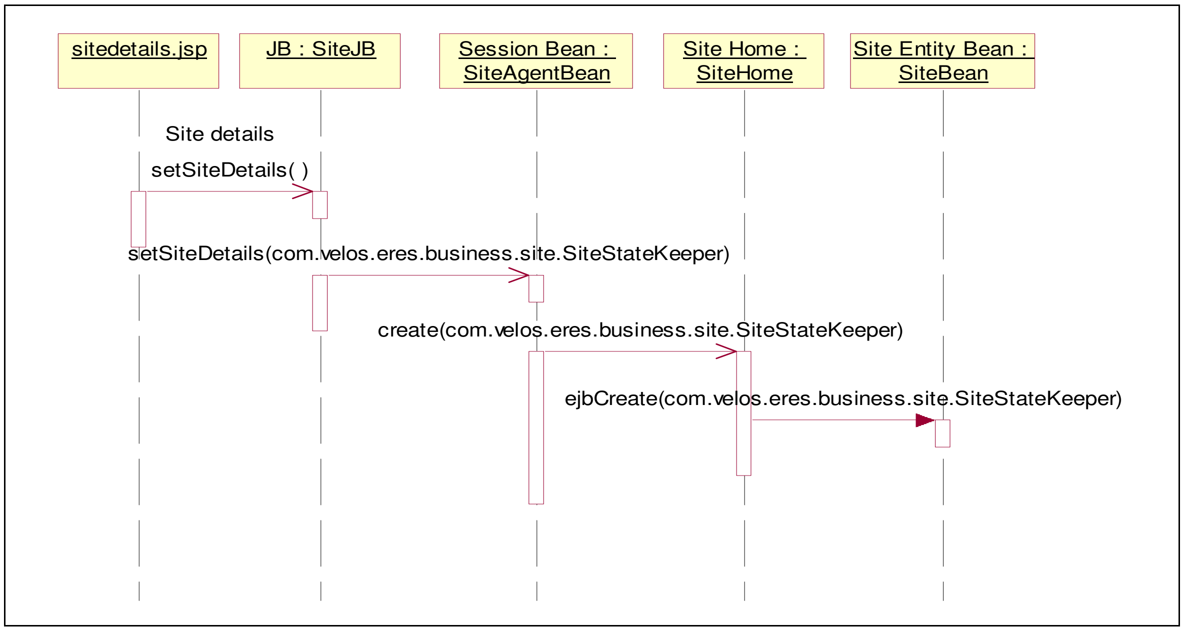
5.2. Login

The login module requires a user to sign on before accessing the Velos eResearch system. It authenticates the user login/password and gives access rights on the basis of his account and group.


ereslogin.jsp
JB : UserJB
Session Bean : UserAgentBean
User Home : UserHome
User Entity Bean : UserRObj
User login/password
validateUser(java.lang.String, java.lang.String)
validateUser(java.lang.String, java.lang.String)
findValidUser(java.lang.String, java.lang.String)
getUserStateKeeper( )
user validated

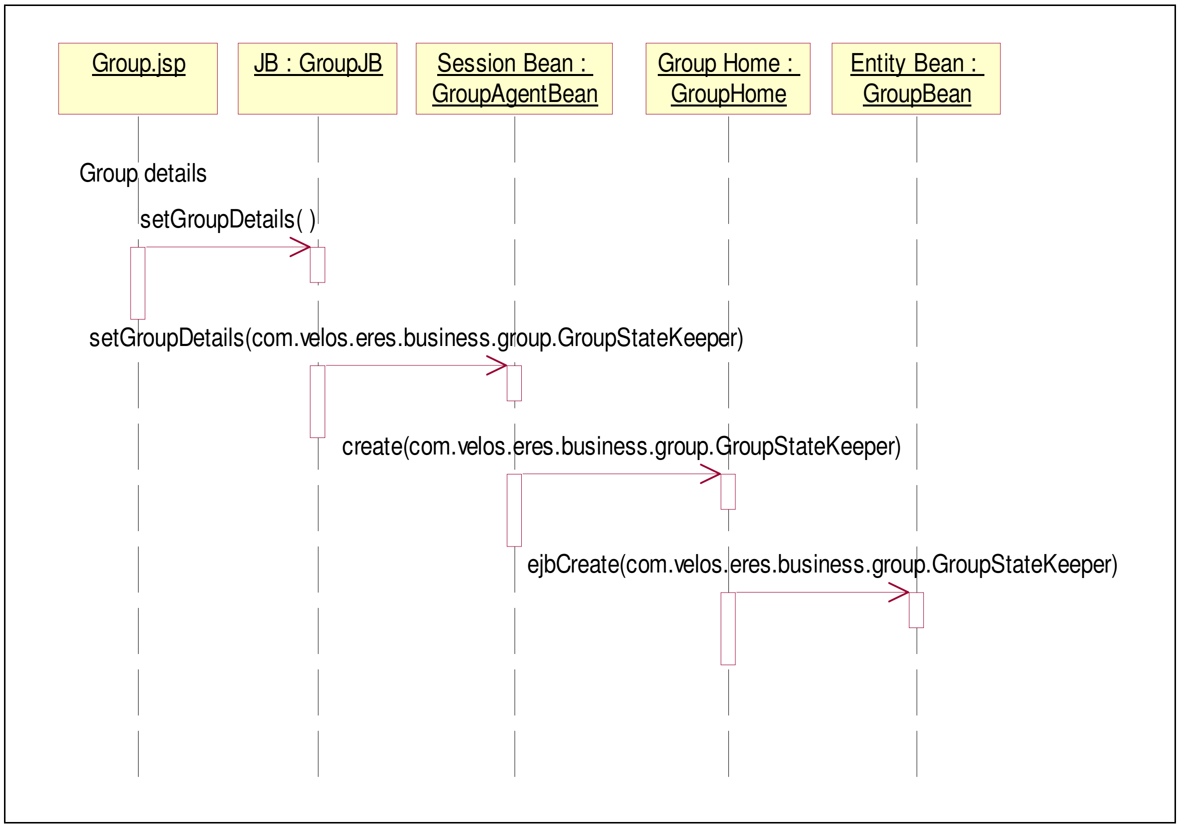
5.3. Organization

Each account can have multiple organizations. The organization module documents the organization information like name, type and address.


sitedetails.jsp
JB : SiteJB
Session Bean : SiteAgentBean
Site Home : SiteHome
Site Entity Bean : SiteBean
Site details
setSiteDetails( )
setSiteDetails(com.velos.eres.business.site.SiteStateKeeper)
create(com.velos.eres.business.site.SiteStateKeeper)
ejbCreate(com.velos.eres.business.site.SiteStateKeeper)

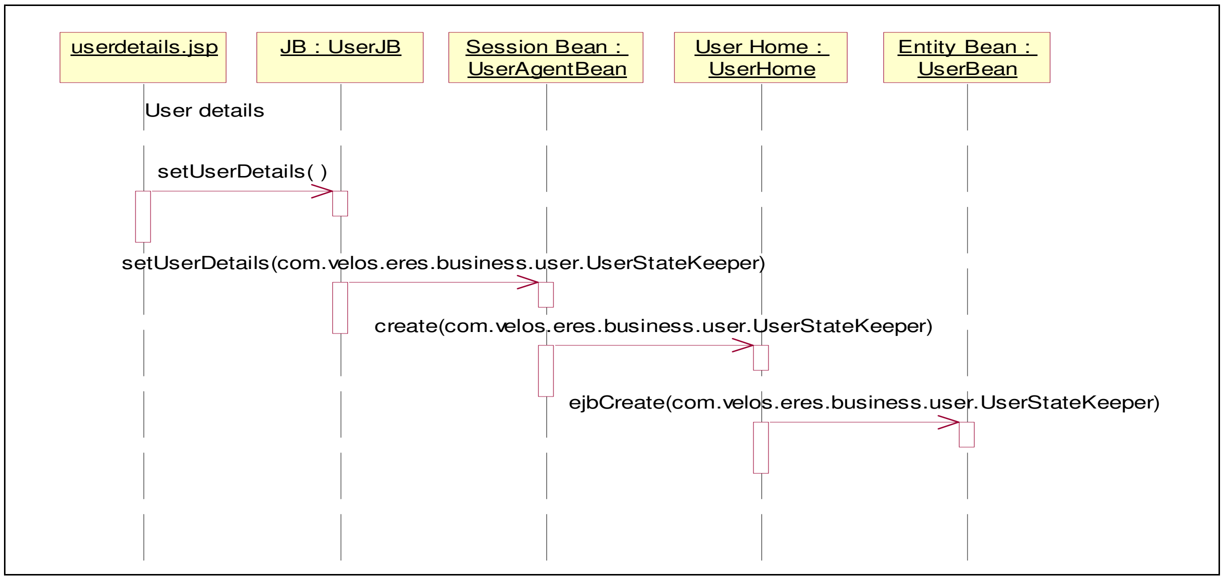
5.4. Group

Each account can have multiple groups. Users having common profile, rights or job type can form a Group. Each group has its own set of access rights. The access rights defined for the groups are applicable to all the users within the group. A user can belong to multiple groups. When the user logs in, his default group governs his access rights for that session.


Group.jsp
JB : GroupJB
Session Bean : GroupAgentBean
Group Home : GroupHome
Entity Bean : GroupBean
Group details
setGroupDetails( )
setGroupDetails(com.velos.eres.business.group.GroupStateKeeper)
create(com.velos.eres.business.group.GroupStateKeeper)
ejbCreate(com.velos.eres.business.group.GroupStateKeeper)

5.5. User

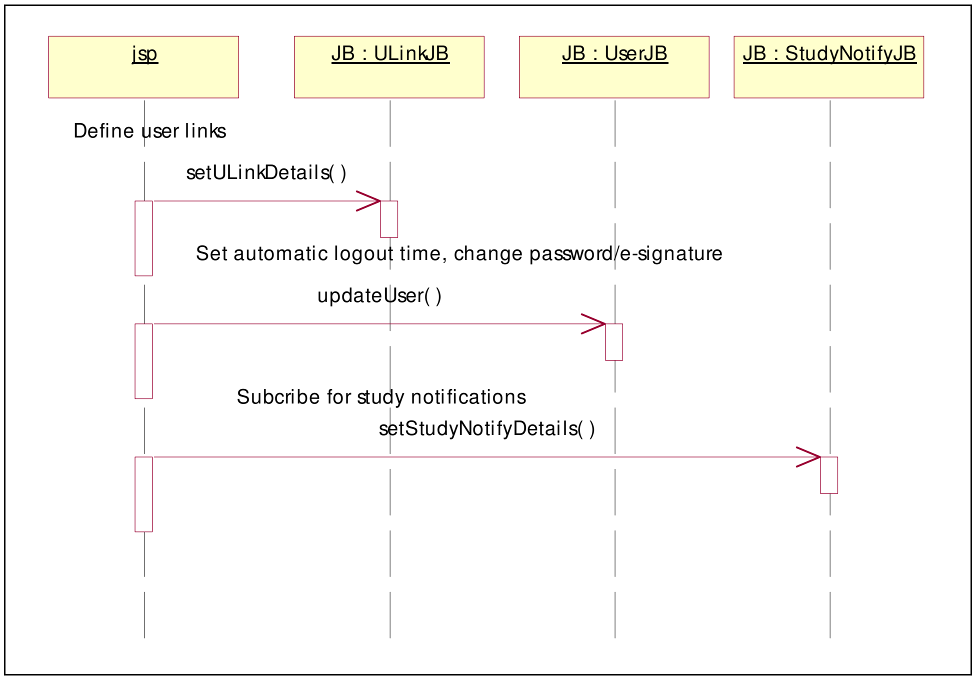
The user module documents user information like name, address, time zone, job profile, login, default group, etc. A sub module – ‘User Groups’ manages association of a user to multiple groups.


userdetails.jsp
JB : UserJB
Session Bean : UserAgentBean
User Home : UserHome
Entity Bean : UserBean
User details
setUserDetails( )
setUserDetails(com.velos.eres.business.user.UserStateKeeper)
create(com.velos.eres.business.user.UserStateKeeper)
ejbCreate(com.velos.eres.business.user.UserStateKeeper)

5.6. *Personalize account*

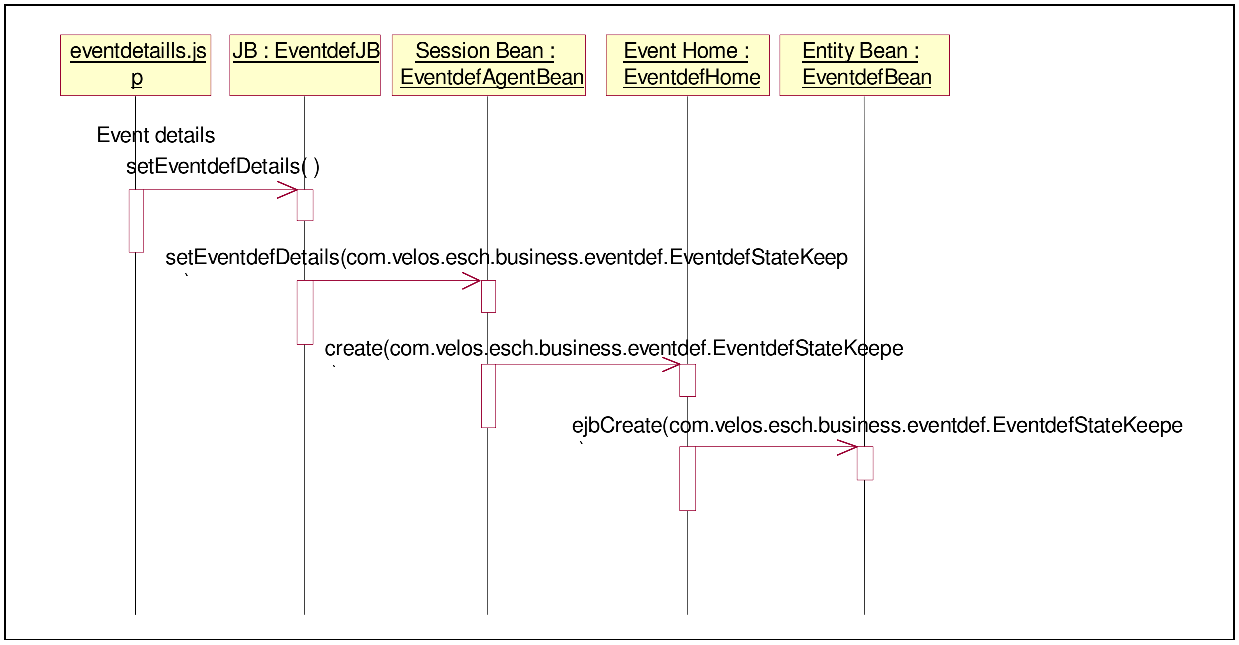
A user can customize his account in the following ways:

- Customizes his own homepage as well as the homepage of all account users (if he has rights) by defining links.
- Subscribes to receive notifications for studies added in specificTherapeutic areas.
- Defines the automatic logout time for his session. The automatic logout feature protects the privacy and security of account.
- Defines and edits the login name, password and e-signature and their expiry time.



5.7. Event Library

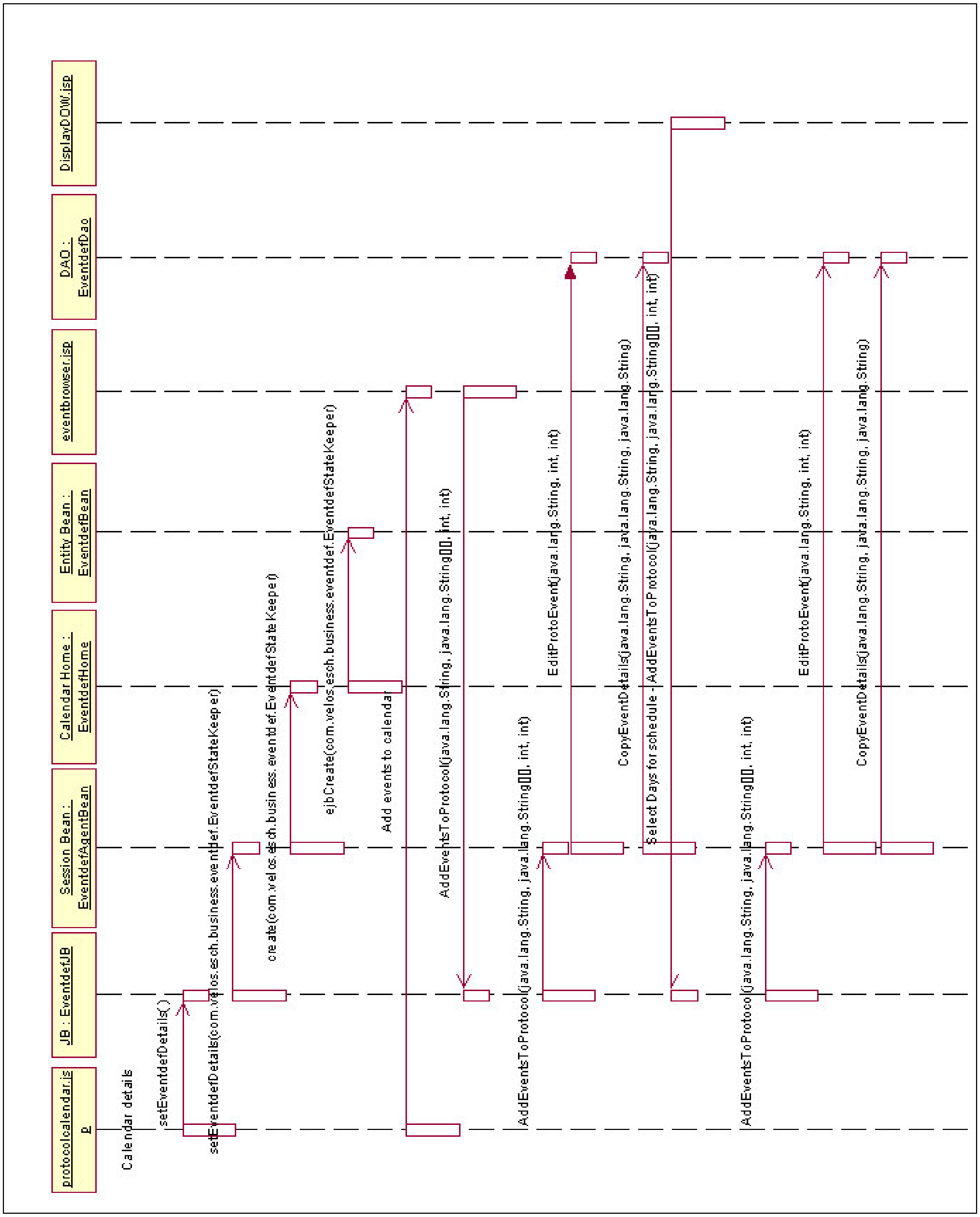
The Event Library is a list of events that are defined by the user and can be used and reused for different protocols with minor or no modifications. The event module documents event information like duration, fuzzy period, linked forms, notifications to be sent, associated cost, resources, CRFs, users, etc.


eventdetaills.jsp
JB : EventdefJB
Session Bean : EventdefAgentBean
Event Home : EventdefHome
Entity Bean : EventdefBean
Event details
setEventdefDetails( )
setEventdefDetails(com.velos.esch.business.eventdef.EventdefStateKeep
create(com.velos.esch.business.eventdef.EventdefStateKeepe
ejbCreate(com.velos.esch.business.eventdef.EventdefStateKeepe

5.8. *Protocol Calendar Library*

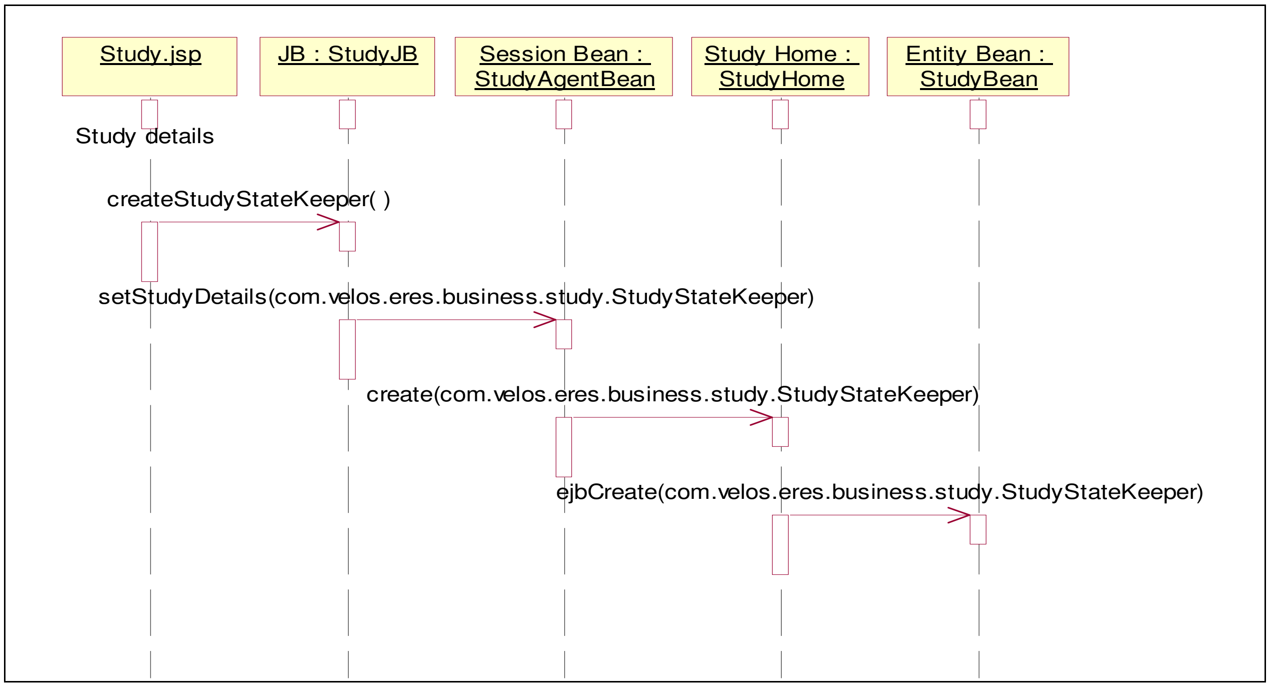
The protocol calendar is a template that defines a hierarchy of events and their intervals. The calendar clearly sets down the events that are to occur, the intervals at which they are to be performed and associated criteria, such as forms to be completed for that event. This portion of the protocol is essential for the actual execution of the trial.

The event library allows the user to easily select events (and to reuse the components for different protocols) from a pre-established list of events. Events selected from the Library and included in the protocol calendar are modifiable to suit the needs of that specific event. Once the calendar has been created, it is available for selection from the protocol calendar library. This template can be used to generate specific schedules for patients simply by selecting the protocol calendar that they are to be associated to and assigning a start date.


protocolcalendar.jsp
JB : EventdefJB
Session Bean : EventdefAgentBean
Calendar Home : EventdefHome
Entity Bean : EventdefBean
eventbrowser.jsp
DAO : EventdefDao
DisplayDOW.jsp
Calendar details
setEventdefDetails( )
setEventdefDetails(com.velos.esch.business.eventdef.EventdefStateKeeper)
create(com.velos.esch.business.eventdef.EventdefStateKeeper)
ejbCreate(com.velos.esch.business.eventdef.EventdefStateKeeper)
Add events to calendar
AddEventsToProtocol(java.lang.String, java.lang.String[], int, int)
AddEventsToProtocol(java.lang.String, java.lang.String[], int, int)
EditProtoEvent(java.lang.String, int, int)
CopyEventDetails(java.lang.String, java.lang.String, java.lang.String, java.lang.String)
Select Days for schedule - AddEventsToProtocol(java.lang.String, java.lang.String[], int, int)
AddEventsToProtocol(java.lang.String, java.lang.String[], int, int)
EditProtoEvent(java.lang.String, int, int)
CopyEventDetails(java.lang.String, java.lang.String, java.lang.String, java.lang.String)

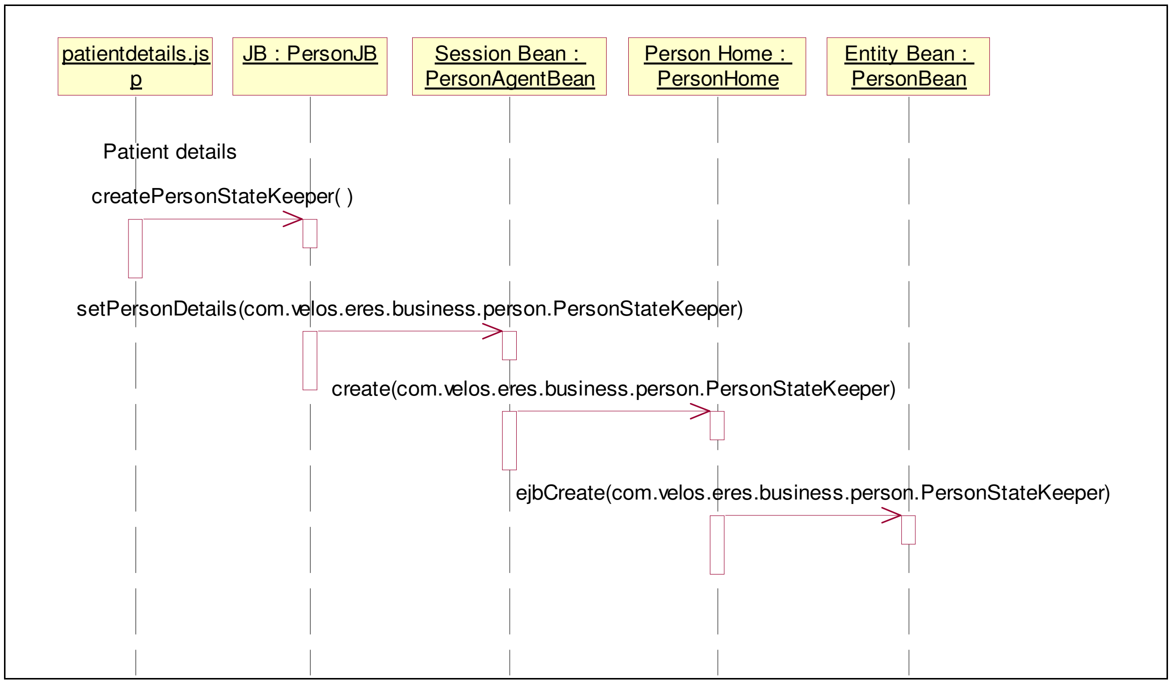
5.9. Study

The study module includes information captured under various sub-modules such as ‘sections’,’team’, ‘calendars’, ‘appendix’, ‘status’ etc.


Study.jsp
JB : StudyJB
Session Bean : StudyAgentBean
Study Home : StudyHome
Entity Bean : StudyBean
Study details
createStudyStateKeeper( )
setStudyDetails(com.velos.eres.business.study.StudyStateKeeper)
create(com.velos.eres.business.study.StudyStateKeeper)
ejbCreate(com.velos.eres.business.study.StudyStateKeeper)

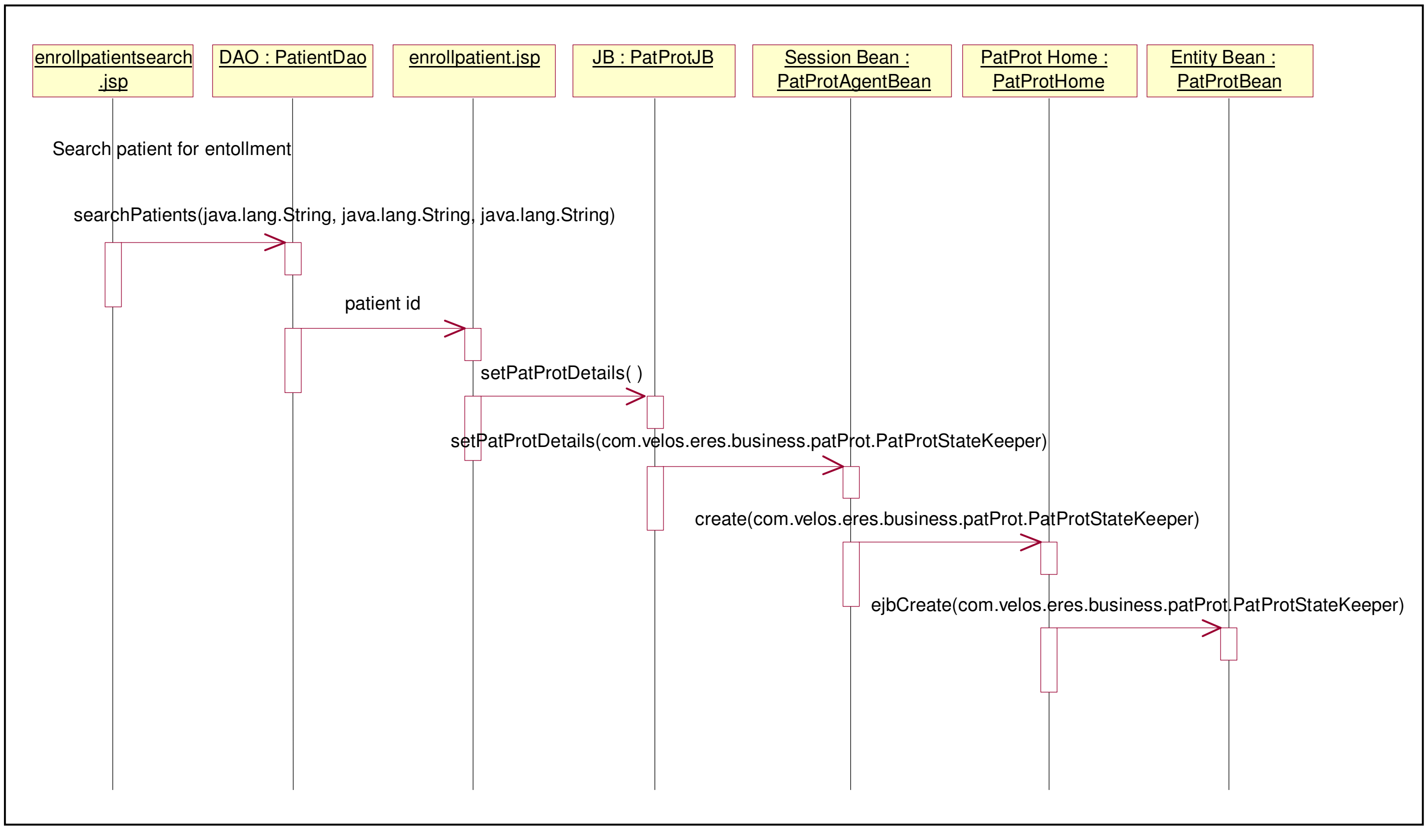
5.10. Patient

The patient module documents patient information such as demographics, registration details, contact information, associated links, forms etc. Patient demographics information is secure and is only accessible by users who have appropriate access rights.


patientdetails.jsp
JB : PersonJB
Session Bean : PersonAgentBean
Person Home : PersonHome
Entity Bean : PersonBean
Patient details
createPersonStateKeeper( )
setPersonDetails(com.velos.eres.business.person.PersonStateKeeper)
create(com.velos.eres.business.person.PersonStateKeeper)
ejbCreate(com.velos.eres.business.person.PersonStateKeeper)

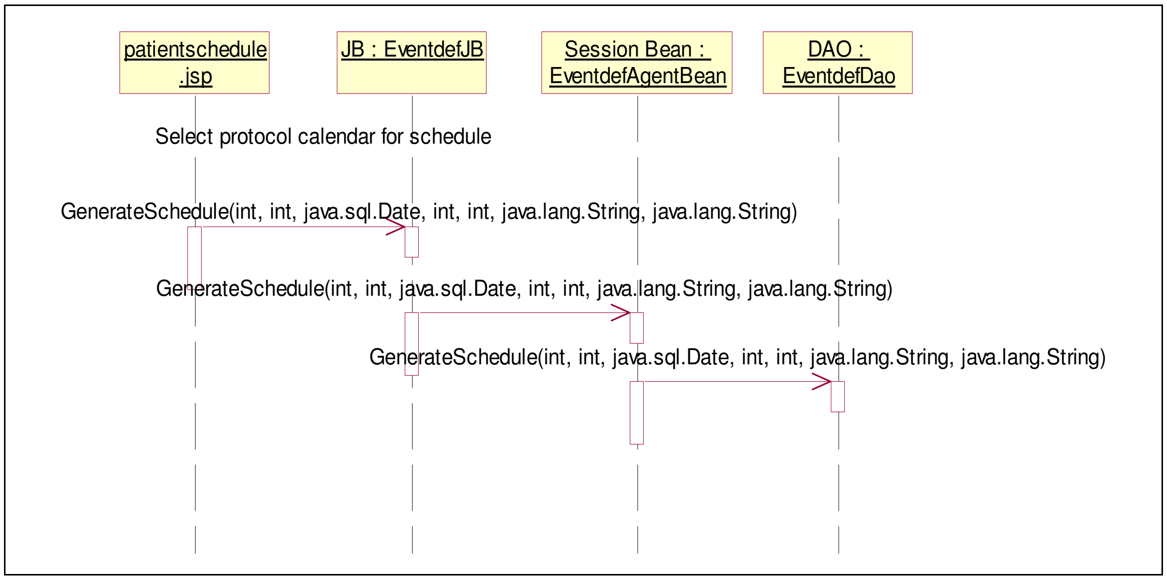
5.10.1. Patient Enrollment

Once the study is active and enrolling, patients can be enrolled and assigned to a specific protocol calendar. A sub – module maintains the Patient Study Status History.


enrollpatientsearch .jsp
DAO : PatientDao
enrollpatient.jsp
JB : PatProtJB
Session Bean : PatProtAgentBean
PatProt Home : PatProtHome
Entity Bean : PatProtBean
Search patient for entollment
searchPatients(java.lang.String, java.lang.String, java.lang.String)
patient id
setPatProtDetails( )
setPatProtDetails(com.velos.eres.business.patProt.PatProtStateKeeper)
create(com.velos.eres.business.patProt.PatProtStateKeeper)
ejbCreate(com.velos.eres.business.patProt.PatProtStateKeeper)

5.10.2. Patient Schedule

Once a patient is enrolled to a study, a schedule can be generated using the calendar template associated to him. The protocol calendar is translated into an accurate patient schedule. It lists the events to be performed and their intervals, tracks the events that have been done, the forms associated with the events, any cost or resource association etc.


patientschedule.jsp
JB : EventdefJB
Session Bean : EventdefAgentBean
DAO : EventdefDao
Select protocol calendar for schedule
GenerateSchedule(int, int, java.sql.Date, int, int, java.lang.String, java.lang.String)
GenerateSchedule(int, int, java.sql.Date, int, int, java.lang.String, java.lang.String)
GenerateSchedule(int, int, java.sql.Date, int, int, java.lang.String, java.lang.String)

5.11. Alerts and Notifications

Once a patient is enrolled and a schedule has been generated, many types of Alerts and Notifications can be sent to the patient / user during study execution. The 'Alerts and Notifications' sub module allows the user to set these notification options for every patient:

5.11.1. Alerts

- Send alert when an event status changes to "Past Scheduled Date"
- Send alert if Serious Adverse Event is reported
- Send alert if Death is reported as Adverse Event outcome

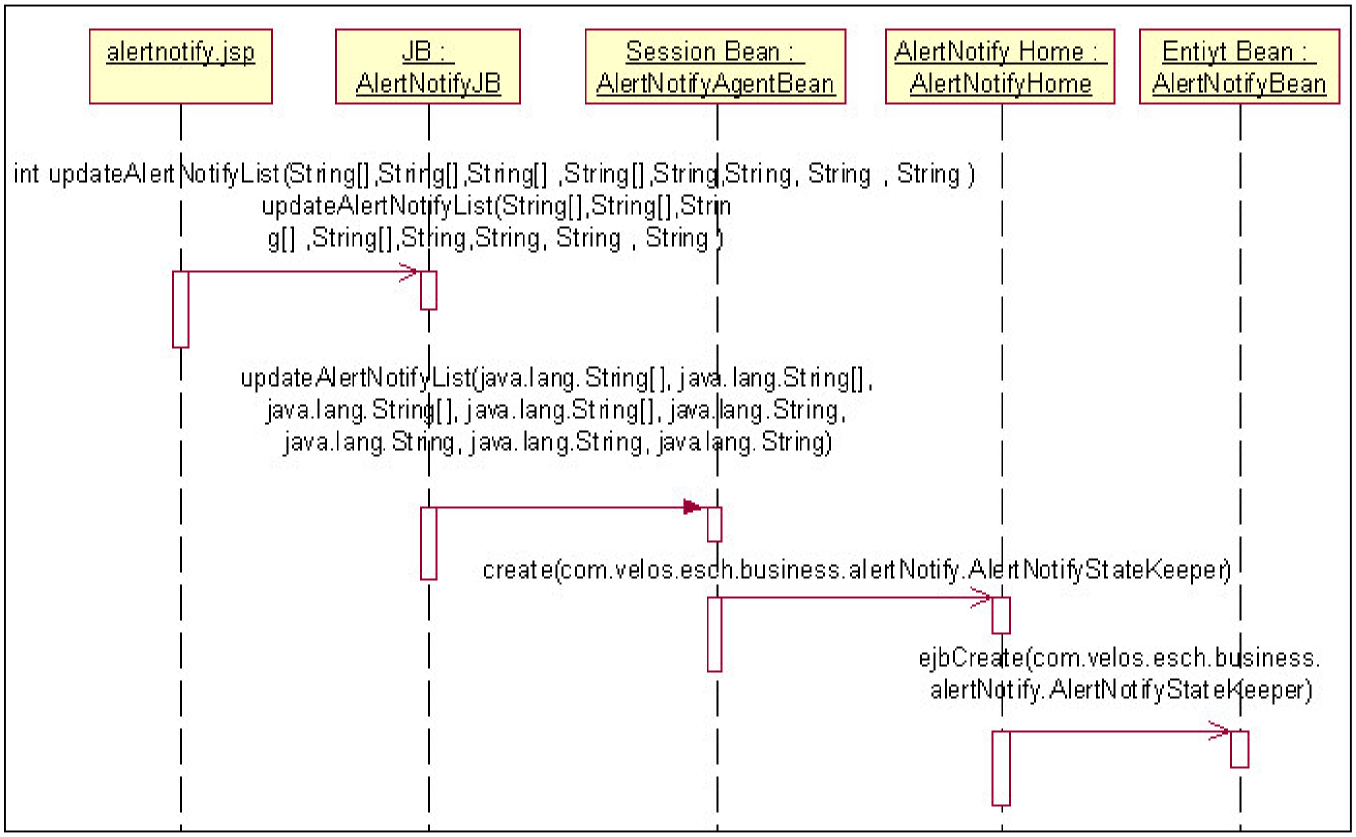
5.11.2. Notifications

- Event Status Change specific Notifications
- CRF Status Change specific Notifications

5.11.3. Options

- Do not send any pre-defined event notifications to this patient
- Do not send any pre-defined event notifications to users (for this patient)

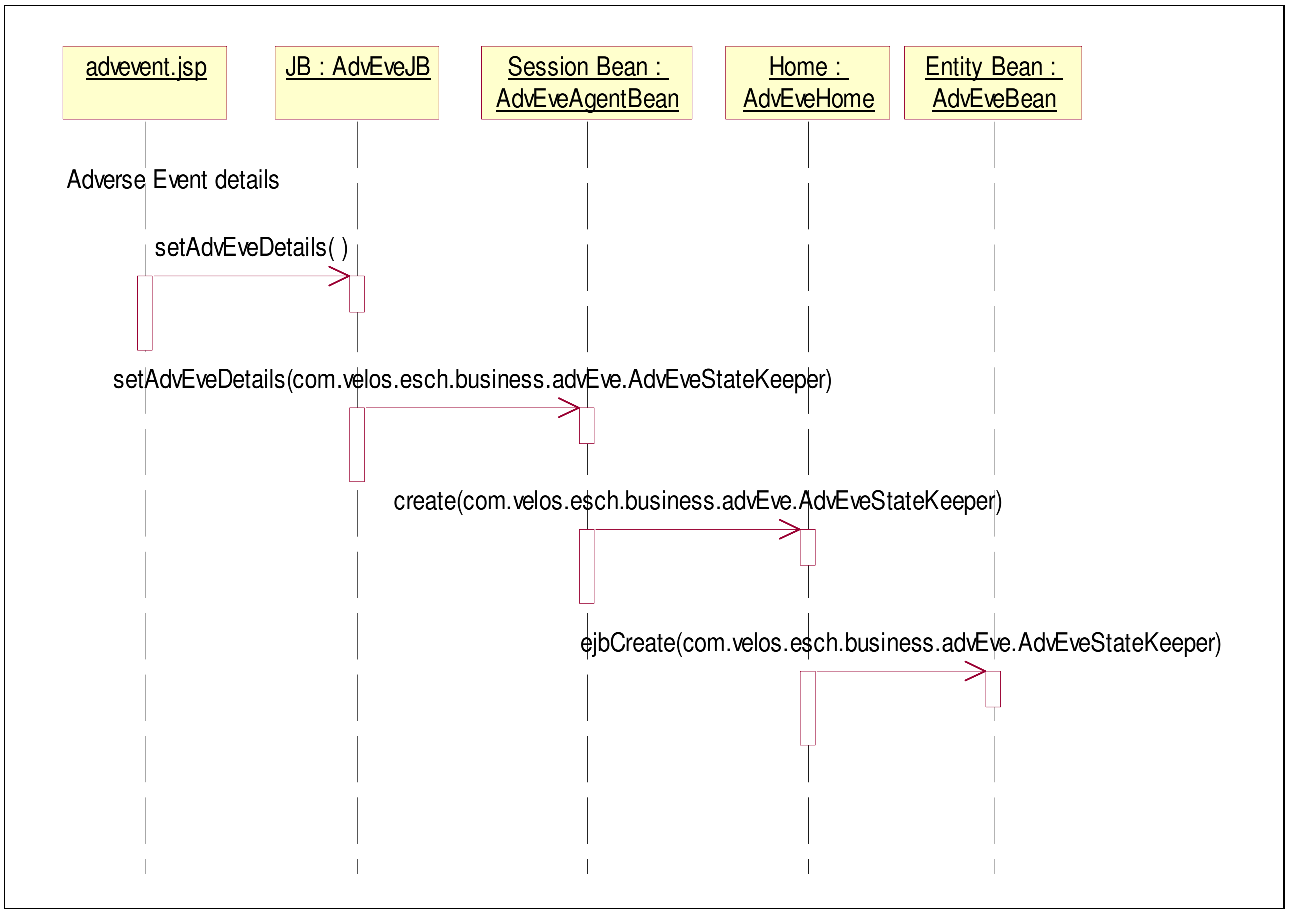
When the above-mentioned Notification options have been defined, notification messages are sent as and when these events occur.


alertnotify.jsp
JB : AlertNotifyJB
Session Bean : AlertNotifyAgentBean
AlertNotify Home : AlertNotifyHome
Entiyt Bean : AlertNotifyBean
int updateAlertNotifyList(String[],String[],String[] ,String[],String,String, String , String)
updateAlertNotifyList(String[],String[],String[] ,String[],String,String, String , String)
updateAlertNotifyList(java.lang.String[], java.lang.String[], java.lang.String[], java.lang.String[], java.lang.String, java.lang.String, java.lang.String, java.lang.String)
create(com.velos.esch.business.alertNotify.AlertNotifyStateKeeper)
ejbCreate(com.velos.esch.business.alertNotify.AlertNotifyStateKeeper)

Another sub module **'Global Alerts and Notifications'** allows the user to set these options for all calendars of a study. This module also gives options such that user can make these settings available for every patient currently enrolled to the study / every patient who will be enrolled in future / revoke the settings for all / revoke for future enrollments.

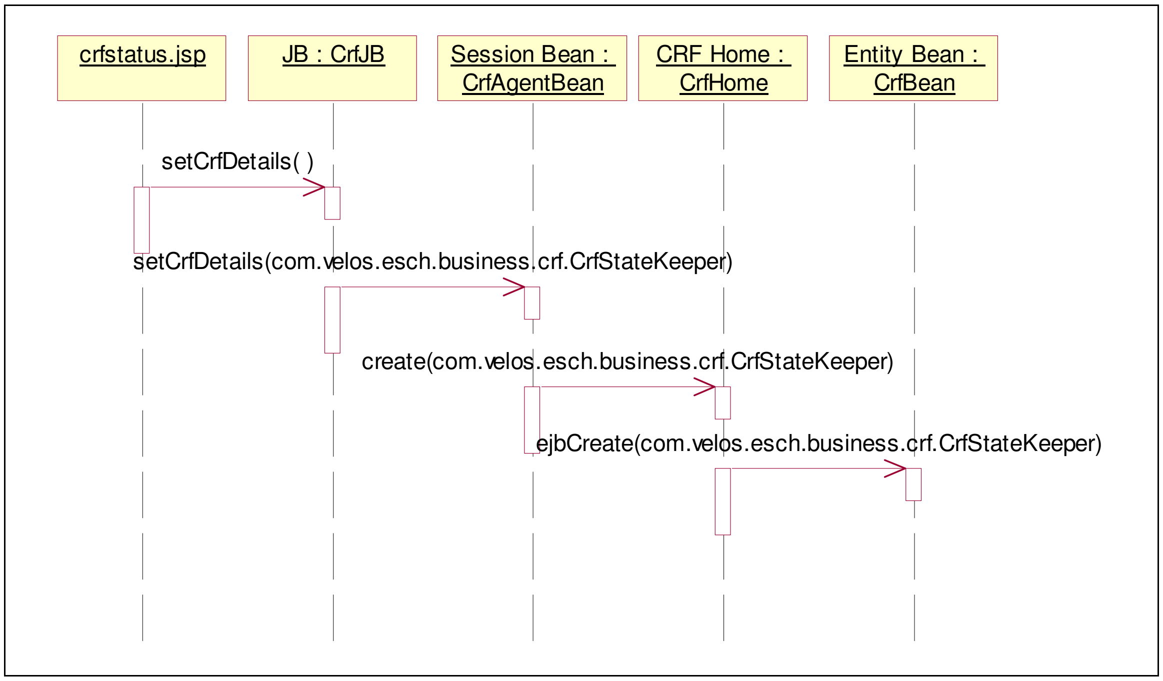
5.11.4. Adverse Events

The 'Adverse Event' sub-module records the 'Adverse Events' that may occur when a patient is enrolled to a study. Details like Adverse Event Type, Start – End Dates, Outcomes, etc are captured. Once the outcomes are recorded, 'Adverse Events' related Notifications are processed according to the patient's or the study's 'Alerts and Notifications' settings.


advevent.jsp
JB : AdvEveJB
Session Bean : AdvEveAgentBean
Home : AdvEveHome
Entity Bean : AdvEveBean
Adverse Event details
setAdvEveDetails( )
setAdvEveDetails(com.velos.esch.business.advEve.AdvEveStateKeeper)
create(com.velos.esch.business.advEve.AdvEveStateKeeper)
ejbCreate(com.velos.esch.business.advEve.AdvEveStateKeeper)

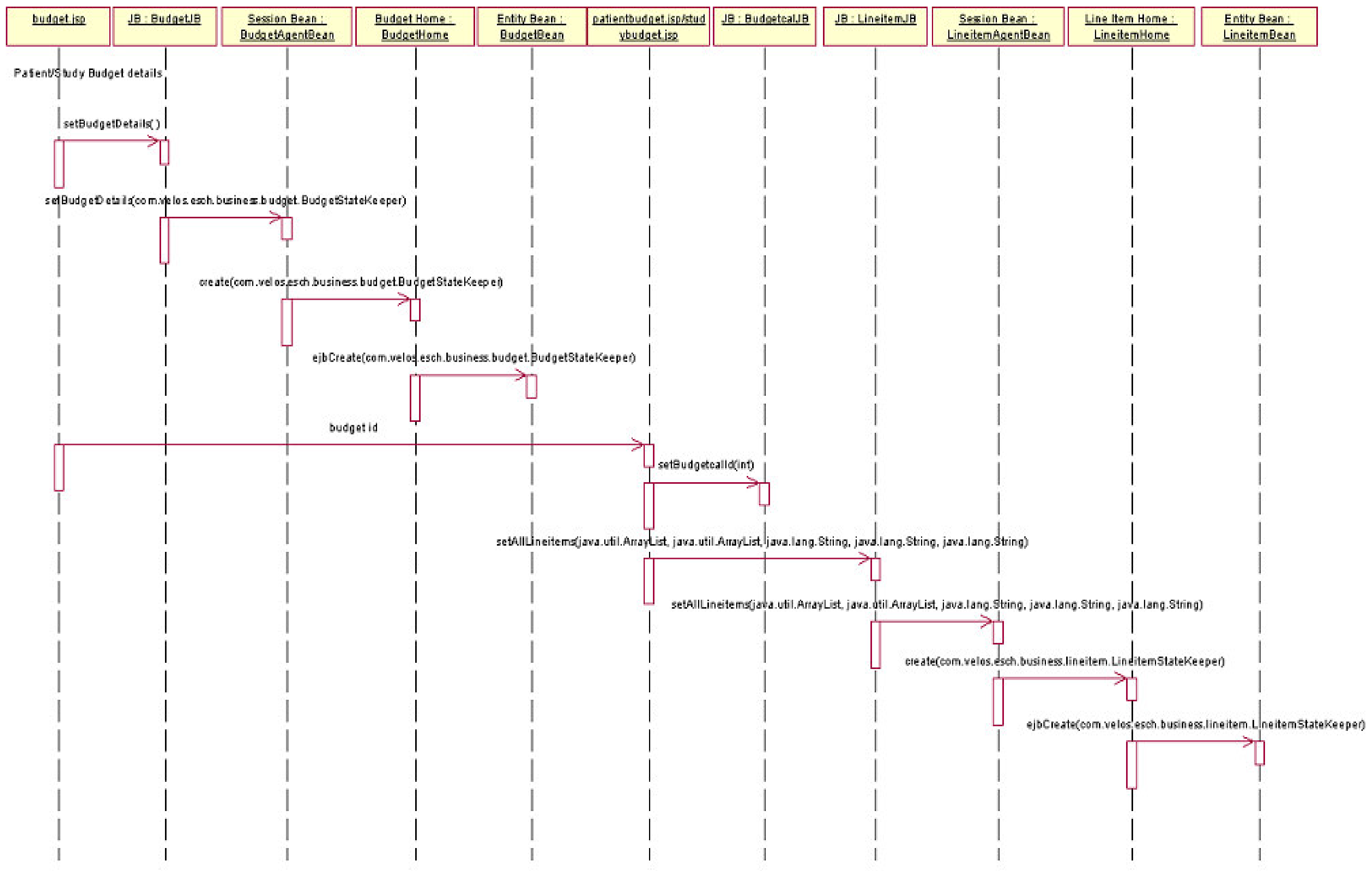
5.12. Patient CRFs and Other Forms

The patient CRF sub module captures CRF related data for each event, such as its status, Review/Approval and Tracking information. If CRF status / or tracking related Notifications have been defined, this information triggers the processing of those Notifications.


crfstatus.jsp
JB : CrfJB
Session Bean : CrfAgentBean
CRF Home : CrfHome
Entity Bean : CrfBean
setCrfDetails( )
setCrfDetails(com.velos.esch.business.crf.CrfStateKeeper)
create(com.velos.esch.business.crf.CrfStateKeeper)
ejbCreate(com.velos.esch.business.crf.CrfStateKeeper)

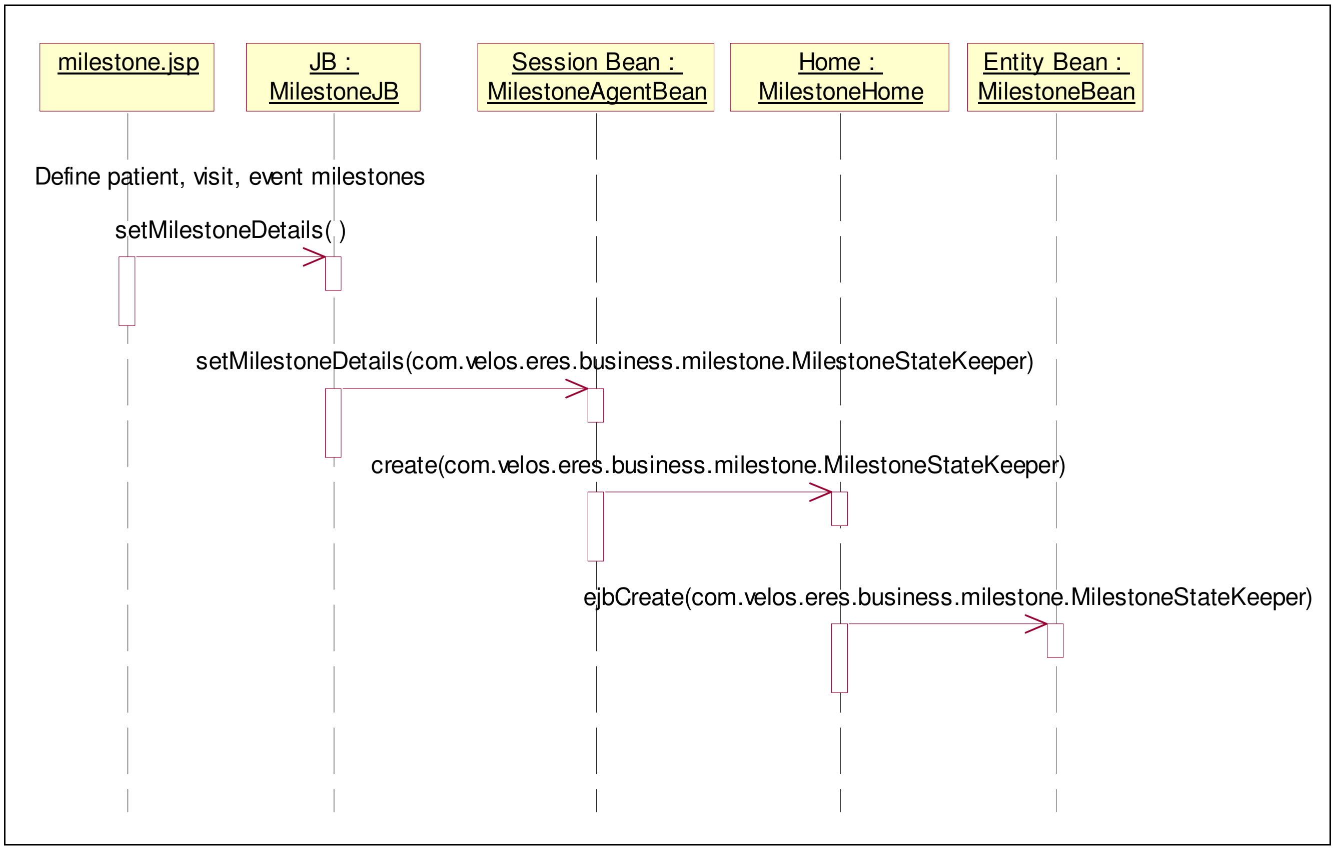
5.13. Budget

The Budget module allows designing and sharing of budget related information. The budget is essential for getting approval for conducting a study, for agreeing on the amounts to be paid and for subsequent billing and cost management of the trial. A Budget can be a ‘patient’ or a ‘study’ budget. It is composed of sub - modules ‘Sections’ and ‘Line Items’. Links and files can be associated with the budget. Access Rights for the budget can be defined at account level, organization level, study team level or explicitly for the user.


budget.jsp
JB : BudgetJB
Session Bean : BudgetAgentBean
Budget Home : BudgetHome
Entity Bean : BudgetBean
patientbudget.jsp/studybudget.jsp
JB : BudgetcalJB
JB : LineitemJB
Session Bean : LineitemAgentBean
Line Item Home : LineitemHome
Entity Bean : LineitemBean
Patient/Study Budget details
setBudgetDetails( )
setBudgetDetails(com.velos.esch.business.budget.BudgetStateKeeper)
create(com.velos.esch.business.budget.BudgetStateKeeper)
ejbCreate(com.velos.esch.business.budget.BudgetStateKeeper)
budget id
setBudgetcalId(int)
setAllLineitems(java.util.ArrayList, java.util.ArrayList, java.lang.String, java.lang.String, java.lang.String)
setAllLineitems(java.util.ArrayList, java.util.ArrayList, java.lang.String, java.lang.String, java.lang.String)
create(com.velos.esch.business.lineitem.LineitemStateKeeper)
ejbCreate(com.velos.esch.business.lineitem.LineitemStateKeeper)

5.14. Milestones

Milestones are major targets within the study that need to be tracked on the basis of their achievement. Milestones can be defined for each study. Notifications are sent when a milestone is achieved (The Milestone Notifications sub module documents the Notification information). Links and files can be associated with Milestones.


milestone.jsp
JB : MilestoneJB
Session Bean : MilestoneAgentBean
Home : MilestoneHome
Entity Bean : MilestoneBean
Define patient, visit, event milestones
setMilestoneDetails( )
setMilestoneDetails(com.velos.eres.business.milestone.MilestoneStateKeeper)
create(com.velos.eres.business.milestone.MilestoneStateKeeper)
ejbCreate(com.velos.eres.business.milestone.MilestoneStateKeeper)

5.15. Reports

Velos eResearch has the powerful ability to aggregate and analyze data that the user has entered into the system and present it in the form of user-friendly reports. These reports can be account related (aggregating and analyzing data across all account users), study related (access controlled analysis of study specific data), patient related or milestones related.

5.15.1. Reports Design

The reports module is composed of several components that work together to create the foundation of the system's reporting capability. These components are:

- Report Sql in database
- Layout Template as XSL in database
- Data Generation in XML
- Report Header/Footer in database
- XSLT Processor - Xalan

Figure 3 below is a diagrammatic representation of the reports architecture.

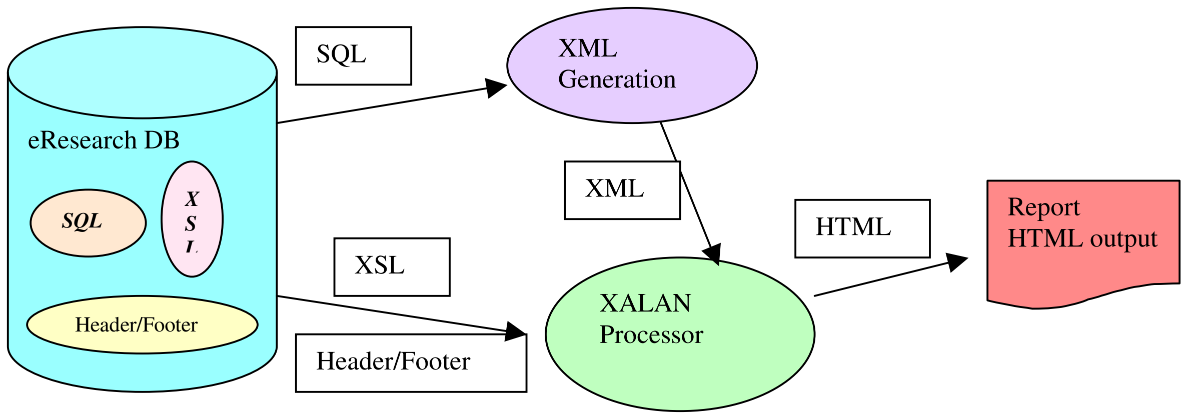


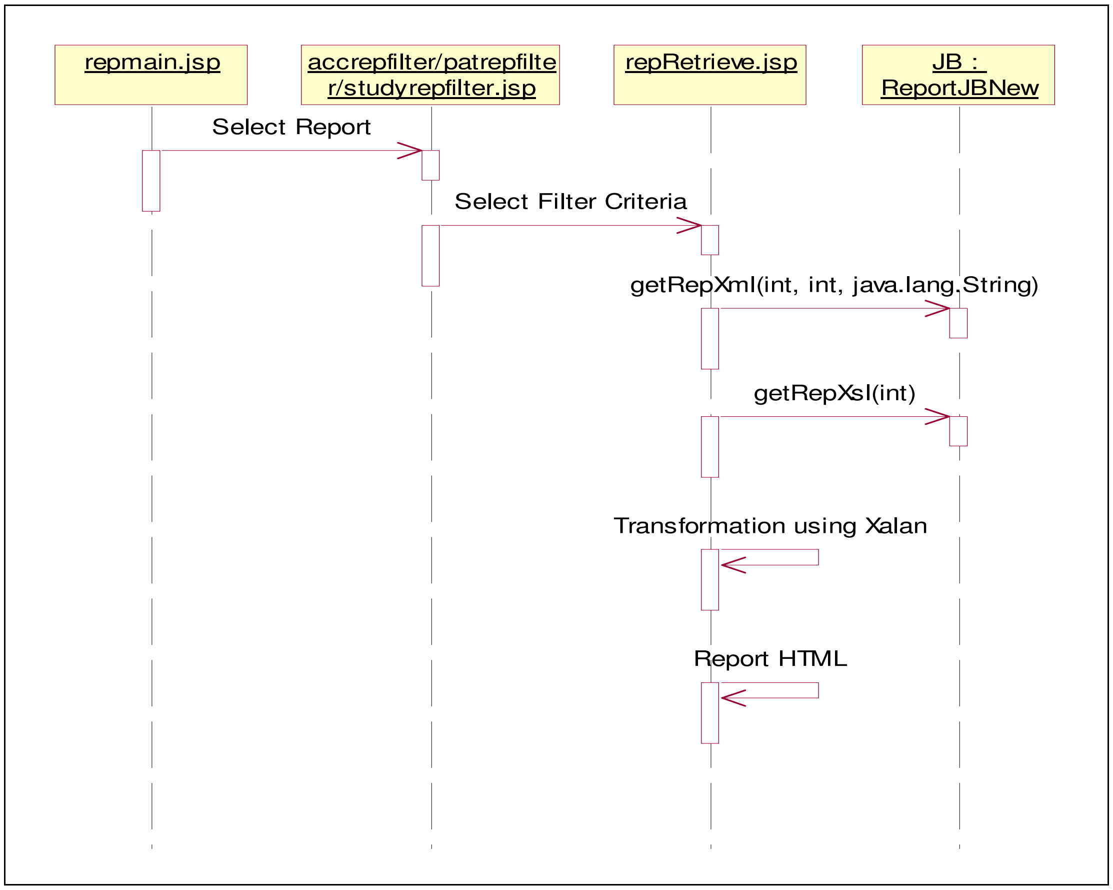
Figure 3: Reports Architecture

The SQL for generating a report is stored in the database. Data is generated by executing SQLs and stored procedures. The generated data is converted into XML. Report layouts are defined in XSL files, also stored in the database. The XML and XSL for each report is fetched from the database and transformed to HTML format using the XSLT Processor – Xalan.

Header and Footer – The header and footer of the reports are also stored in the database. This provides an easy option for customizing headers/footers for reports for specific reports or account.

Saving reports in other formats – This module provides an easy way to download the reports in Word, Excel or HTML Format. The user can then customize the report to suit his needs by inserting titles, formatting the text, applying filters, creating charts and graphs or any other feature of the application to which the data is downloaded.

The figure below depicts the interaction diagram of report JSP and its beans.


repmain.jsp
accrepfilter/patrepfilter/studyrepfilter.jsp
repRetrieve.jsp
JB : ReportJBNew
Select Report
Select Filter Criteria
getRepXml(int, int, java.lang.String)
getRepXsl(int)
Transformation using Xalan
Report HTML

5.15.2. Advantage

The report presentation and the data are two separate entities. This has the advantage of being able to change the report layout at any point of time without affecting other areas.

5.15.3. Reports available in Velos eResearch

- All Account Users
- Patient Events (Done)
- Patient Events (To Be Done)
- All Patients Event List
- Patient Schedule

- Summary for Public Broadcast
- Estimated Study Budget
- All Patients Enrolled to a Study
- Active Protocols
- User Profile
- Study Team
- Protocols by User
- Protocols By Therapeutic Area
- Public Protocols
- Protocol Tracking
- Protocols by Organization
- Complete Protocol
- Assigned External Users
- Protocol Calendar Template
- Patients Associated Cost
- Account Report (Detailed)
- Account Report (Summary)
- Patient CRF Tracking
- All Patients Associated Cost
- Patient Event/CRF Status
- Milestones Summary (All Patients)
- Milestones Summary (Patient)
- Milestones by Month
- Milestones by Quarter
- Milestones by Year
- Payments Received by Month
- Milestones Summary (Study)
- Payments Received by Quarter
- Payments Received by Year
- Milestones/Payment Discrepancy by Month
- Milestones/Payment Discrepancy by Quarter
- Milestones/Payment Discrepancy by Year
- Account Report (Detailed)
- Account Report (Summary)
- Audit Trail reports

6. Security in Velos eResearch

Apart from using 128-encryption to protect sensitive information online, Velos eResearch also has application level security. Each user has a set of rights for accessing different modules. The various levels at which application rights can be defined are:

- **Account level security** – Velos administrator reserves the rights for granting access permission to an account for these modules:
  - Enhanced Patient Tracking
  - Budgeting
  - Milestones

These rights are applicable to all users within the account. The modules are either enabled/disabled and are made visible/invisible in the left menu panel according to the rights of the logged in user.

- **Login security** – The user login name and password are authenticated and verified at the time of login. When the user logins, he will be given access rights according to his default group's access rights (explained below).
-
- **Modular security** – The Account administrator can provide 'New', 'Edit' and 'View' rights for different modules. A user with 'New' right has the permission to add/create information in the system. A user with 'Edit' right cannot create but can modify the already entered data. A user with 'View' permission can only view the data.
-
- **Group level security** –Group administrators can define 'New', 'Edit' and 'View' rights for each module. All users within a group have the same set of rights. The modules for which rights can be defined are:
  - Study Protocol
  - Manage Groups
  - Manage Users
  - Assign Users to Group
  - Manage Group Rights
  - Manage Organizations
  - Manage Account Specific Links
  - Protocol Library
  - Reports
  - Manage Patients
  - Budgeting

- **Study level security** – Study data managers can define ‘New’, ‘Edit’ and ‘View’ rights for accessing study details. Each study team member has his own set of rights for accessing the study. External account users as well as internal account users can be provided with a modifiable or viewable copy of protocol. Access rights can be defined for:
  - Study Status
  - Study Team
  - Associate External Users
  - Study Summary
  - Study Sections
  - Appendix
  - Study Notifications
  - Manage Patients
  - Edit / View Complete Patient Data
  - Protocol Calendar
  - View Study Reports
  - Milestones
  - Create a Version

- **Budget level security** - The ‘creator’ of a budget can define access rights for accessing details about the budget and appendix. Budget rights can be defined at four levels:
  - all users of the account – if budget rights are defined for an account, they are applicable to all the users within that account
  - all users of the organization – if budget rights are defined for an organization, they are applicable to all the users within that organization
  - all users within the study team – if budget rights are defined for a study team, they are applicable to all study team members of the specified study
  - individual user – if budget rights are defined for an individual user, these rights supercede rights defined at account/organization or study team level

7. The Velos eResearch Mailer

The "eResearch mailer" is a mailing service that runs in addition to the application servers. The Velos eResearch system provides various alerts and notification services. Some of the notifications that a user can configure from various modules of the application are:

- Patient's Event status change
- Patient's Event Schedule
- "Past Scheduled Date" - when an event passes its scheduled date and is not yet processed
- Patient's CRF/Other Forms Tracking `
- Patient's CRF/Other Forms status Change
- Milestone Achievement
- Alerts for Adverse Events
- Milestone reports' completion

These notifications can be received as:

- Information specific individual mail
- Clubbed Notification - one single mail comprising of different types of messages
- Small message for pager/cell devices
-

7.1. *How it works*

- Mail messages are created when:
    - A particular type of information is saved and messages related to it are generated (using triggers) and inserted in the table **sch_dispatchmsg (user sch)** immediately. Notifications related to Adverse Events (serious adverse event, 'death' as outcome), Event Status change, CRF status change, CRF tracking (when message is to be sent on same date), Milestone Report Generation etc. fall in this category.
    - A patient's schedule is generated and the pre-defined event based messages (for patient as well as user) are generated and dumped in **sch_msgtran (user sch)**. These messages can be post dated too but the mailer sends them on their respective scheduled dates.
    - The database job for executing **SP_GENCLUBMAIL** (from package **PKG_CLUB , user sch**) is run. This procedure:
        - Executes the procedures for CRF Tracking, Past Scheduled Date and Milestone Achievement Notification procedures. These procedures work according to

existing data as well as user's settings for alerts and notifications. The generated messages are inserted in the table **sch_msgtran**.

- The next step clubs all such messages for a user and date combination and transfers the information to the table **sch_dispatchmsg**

- Table **sch_msgtext** stores the message templates used for the above mentioned notifications

- The mailer is activated at an interval of 1 minute and 'looks' for pending mail messages in the table **sch_dispatchmsg**, creates the mail message and sends it to the intended recipient.

- The mailer works according to Timezones. It uses the Timezone information specified for users and patients. E.g. if the mail service is running in Fremont, and the mail recipient is in Japan, in this case the mail is sent according to Japan Timezone and not Pacific Standard Time.

- The "eResearch mailer" should run without interruptions to send notifications in a timely manner.

7.2. Setting up the eResearch mailer

eresMailer.ini - eresMailer.ini stores configuration for the following:

- Database connection information
- SMTP host

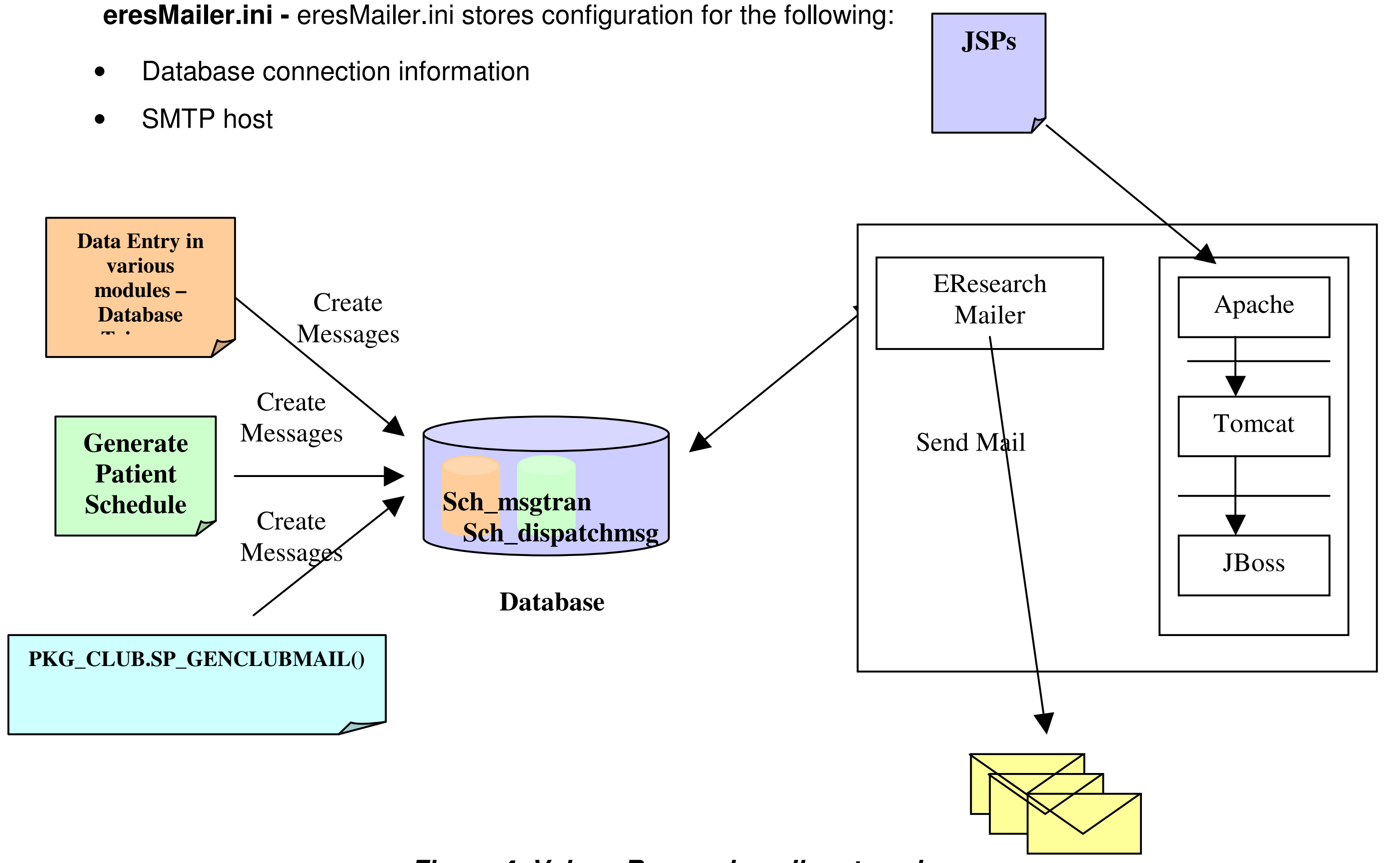


Figure 4: Velos eResearch mailer at work

8. Audit Trail in Velos eResearch

The Audit Trail is an important security component of the Velos eResearch application. The module keeps a record of users accessing the system and operations performed by them during a given period of time. It enables the Velos administrator to track all user movements and updates throughout the Velos eResearch application. The audit trail includes:

- Records that the transaction has occurred
- Tracking Insert operation
- Tracking Update operation with record of OLD and NEW values
- Tracking Delete operation

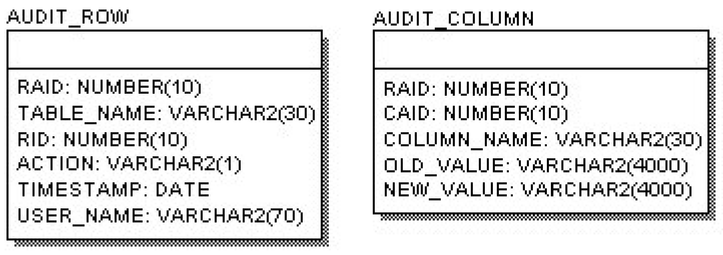
8.1. Design

Each table that has to be audited has six columns. These columns are:

- RID - The Row id uniquely identifies the row in the database. This is populated from a sequence and is unique throughout the database.
- CREATOR - The id of the user who created the record in the Velos eResearch system.
- LAST_MODIFIED_BY - The id of the user who last modified the row in the table.
- LAST_MODIFIED_DATE - Date and time of record modification.
- CREATED_ON - Date and time of record creation.
- IP_ADD - Stores the IP address of the client machine.

8.2. Tables

- AUDIT_ROW - Stores the table, action, date-time, user and row id of the row on which the operation is being done.
- AUDIT_COLUMN - Stores the information about the column with its old and new values and has a reference to the AUDIT_ROW table through the Record Id of the modified row.
- AUDIT_DELETE - Stores the complete information along with the values of the deleted records.



The values of the deleted row are concatenated and stored as one string. Blob columns are not stored.

8.3. Triggers

The following triggers are created for each table:

- **Before insert Trigger** to record the insertion of a new row. It records the event in AUDIT_ROW table.
- **After update Trigger** is designed to record the update in the column values of a row. This trigger first inserts a record in AUDIT_ROW table with the Record Id of the updated row and then writes the old and new values of the updated columns in AUDIT_COLUMN table.
- **After delete Trigger** is designed to record the deletion of a row. This trigger first records the event in AUDIT_ROW table and then writes the concatenated values of the row in AUDIT_DELETE table.

8.4. Package and Stored Procedures

AUDIT_TRAIL - This package has two stored procedures in it. The triggers use these procedures for recording the transactions and updates.

- RECORD_TRANSCATION - Inserts a record to AUDIT_ROW table to mark a transaction -can be a delete, insert or an update.
- COLUMN_UPDATE - Inserts record(s) in AUDIT_COLUMN table for each modified column.

<u>Two Sequences</u>

- SEQ_RID - this sequence is used to get the value of RID column in the Audited table(s). A new value is extracted from the sequence every time a record is inserted in any of the audited tables.
- SEQ_AUDIT - this sequence generates unique transaction numbers.

8.5. How The Audit Trail Works

This approach is based on the use of triggers for automatic recording of change in records.

- When a new record is inserted in the audited table, the BEFORE_INSERT trigger updates 4 columns in the table. The other columns are updated from the application. An entry goes to the AUDIT_ROW table.
- Whenever a modification is made to the record, an entry is added to the AUDIT_ROW and AUDIT_COLUMN tables. In the AUDIT_ROW table, one record is inserted per transaction while in the AUDIT_COLUMN table, a record is inserted for each modified column per transaction.

- When a record is deleted from the audited table, a record is inserted in AUDIT_ROW to indicate the deletion of record, and another record is added to AUDIT_DELETE table where the complete deleted record is stored as PIPE separated values.

8.6. Audit Reports

Audit reports can be viewed on the basis of:

- Dates
- Action and Dates
- Action, User and Dates
- User and Dates

The stored procedure SP_AUDREP generates an XML from SQLs for the Audit module. The SQLs are stored in the AUDIT_REP.AUD_SQL column. The respective XSL for the report is fetched from the database. XSLT processor – Xalan transforms the XML and XSL to HTML and generates the audit report.